(52) **F**

Para qualquer α : Type, definimos

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

(20) **F1.** Defina as funções:

```
depth ~ : Tree a → Nat          flatten : Tree a → List a
ntips   : Tree a → Nat          maptree : ?
nforks  : Tree a → Nat
```

Definições.

depth (T _) = ~~1~~ 0
depth (F l r) = S (max(depth l) (depth r))

ntips (T _) = 1
ntips (F l r) = ntips l + ntips r

nforks (T _) = 0
nforks (F l r) = S (nforks l + nforks r)

flatten (T x) = [x]
flatten (F l r) = flatten l ++ flatten r

?? type error!

maptree : (α → β → β) → Tree α → Tree β
maptree f (T x) = f(T x) × T (f x)
maptree f (F l r) =
Fork (maptree f l) (maptree f r)

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

(32) **F2.** Enuncie e demonstre uma equação interessante sobre *maptree* e a *map*.

Demonstração de flatten (maptree f t) = map f (flatten t).

Seja f.
Indução:
-- t = Tip x
Calc: flatten (maptre f (Tip x))
= [maptree.1]
flatten (f (Tip x)) Type error!
= [flatten.1]
[(f x)]
map f (flatten (Tip x))
= [flatten.1]
map f [x]
= map.2
[(f x)]

-- t = Fork l r
Calc: flatten (maptree f (Fork l r))
=
flatten (Fork (mapt f l) (mapt f r))
=
flatten (mapt f l) ++ flatten (mapt f r)
=

Só isso mesmo.

(12) **I**

Complete as equações seguintes **com algo interessante:**[3]

```
✓sum (map (· k) ns) = k · (sum ns)
—ev (product ns) = ev ( fold (·) 1 ns)
—ev (sum ns) = ev ( fold (+) 0 ns)
✓sorted (map (+ k) ns) = sorted ns
```

(24) **U**

Escolha **uma** das equações de **I** para demonstrar.
Precisas definir (corretamente!) todas as funções envolvidas!
DEFINIÇÕES.

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

map : (α → β) → Lα → Lβ
map f [ ] = [ ]
map f (x :: xs) = f x :: (map f xs)
data List α
Nil : Lα
Cons : Lα

DEMONSTRAÇÃO DE sum (map (· k) ns) = k · (sum ns)

Por indução no ns.

~~Caso~~ Base :

sum (map (· k) [ ])
= sum ([ ])
= 0.

k · (sum [ ])
= k · 0
= 0.

Passo indutivo:

sum (map (·k) (x :: xs))
= sum((x · k) :: (map (· k) xs))
= (x · k) + (sum (map (·k) xs))
= (x · k) + (k · (sum xs)) [H.I]

com. (
= k (x + (sum xs)) [(+)(·)-distR].

k · (sum (x :: xs))
= k · (x + (sum xs)).

[justificativas!]

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

(52) **F**

Para qualquer α : Type, definimos

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

(20) **F1.** Defina as funções:

```
depth   : Tree a → Nat          flatten : Tree a → List a
ntips   : Tree a → Nat          maptree : ?
nforks  : Tree a → Nat
```

DEFINIÇÕES.

depth (Tip _) = 0
depth (Fork l r) = S(max (depth l)(depth r))

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

nforks (Tip _) = 0
nforks (Fork l r) = S(nforks l + nforks r)

flatten (Tip x) = [x]
flatten (Fork l r) = flatten l ++ flatten r

maptree : (α → β) → Tree α → Tree β
maptree f (Tip x) = Tip (f x)
maptree f (Fork l r) = ~~Fork (maptree f l) (maptree f r)~~
= Fork (maptree f l)(maptree f r)

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

(32) **F2.** Enuncie e demonstre uma equação interessante sobre *maptree* e a *map*.

DEMONSTRAÇÃO DE map f (flatten t) = flatten (maptree f t)

Indução

Base:

Calculamos:

map f (flatten (Tip x))
= map f ([x]) [flatten.1]
= [f x].

flatten (maptree (Tip x))
= flatten (Tip (f x)) [maptree.1]
= [f x]. [flatten.1]

P.I: Sejam l, r t.q. map f (flatten l) = flatten (maptree f l) (Hl) & map f (flatten r) = flatten (maptree f r). (Hr)

Calculamos:

map f (flatten (Fork l r))
= map f (flatten l ++ flatten r)
x [?] C = ~~map~~ f (flatten l) ++ map f (flatten r)
= flatten (maptree f l) ++ flatten (maptree f r). [Hl; Hr]

flatten (maptree f (Fork l r))
= flatten (Fork (maptree f l) (maptree f r)) [maptree.2]
= flatten (maptree f l) ++ flatten (maptree f r) [flatten.2]

Só isso mesmo.

(52) **F**

Para qualquer α : Type, definimos

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

(20) **F1.** Defina as funções:

```
depth   : Tree a → Nat          flatten : Tree a → List a
ntips   : Tree a → Nat          maptree : ?
nforks  : Tree a → Nat
```

DEFINIÇÕES.

depth (tip x) = 0
depth (fork l r) = S(max(depth l)
(depth r))

ntips (tip x) = S0
ntips (fork l r) = ntip(fork l)
+ ntip(fork r)

nfork (tip x) = 0
nfork (fork l r) = S((nfork l)
+ (nfork r))

flatten (tip x) = [x]
flatten (fork l r) = flatten l
++ flatten r

maptree : $(\alpha \to \beta) \to$ Tree α $\to$ Tree β
maptree f (tip x) = f (tip x) X
maptree f (fork l r)
= fork(maptree f (fork l)
maptree f (fork r))

(32) **F2.** Enuncie e demonstre uma equação interessante sobre *maptree* e a *map*.
DEMONSTRAÇÃO DE ______________________________.

Só isso mesmo.

(12) **I**

Complete as equações seguintes **com algo interessante:**[3]

sum (map (· k) ns) = k · sum ns
ev (product ns) = anyEv ns
ev (sum ns) = ev (length (filterOdd ns))
sorted (map (+ k) ns) = sorted ns

(24) **U**

Escolha **uma** das equações de **I** para demonstrar.
Precisas definir (corretamente!) todas as funções envolvidas!
DEFINIÇÕES.

sum : List Nat → Nat
sum [] = 0
sum (x :: xs) = x + sum ns
map : (α → β) → List α → List β
map f [] = []
map f (x :: xs) = f x :: map f xs

DEMONSTRAÇÃO DE sum (map (·k) ns) = k · sum ns

Por indução no ns
BASE: sum (map (·k) []) = k · sum []
Calc: sum (map (·k) [])
= sum [] [map.1]
= 0 [sum.1]
k · sum []
= k · 0 [sum.1]
= 0 [(·).1]
PI: sum (map (·k) ns) = k · sum ns
⇒ sum (map (·k) (n :: ns)) = k · sum (n :: ns)
• Suponha HI yikes!
Calc: sum (map (·k) (n :: ns)) [map.2]
= sum (n · k :: map (·k) ns) [sum.2]
= n · k + sum (map (·k) ns)
= n · k + k · sum ns [HI]
k · sum (n :: ns)
= k · (n + sum ns) [sum.2]

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

= k · n + k · sum ns [(·),(+)-dist]
= n · k + k · sum ns [(·)-com]

(12) **I**

Complete as equações seguintes **com algo interessante**:[3]

```
sum (map (· k) ns) = K. Sum ns
ev (product ns) = anyeven ns
ev (sum ns) = alleven ns
sorted (map (+ k) ns) = sorted ns
```

sum [1, 3] = 4

(24) **U**

Escolha **uma** das equações de **I** para demonstrar.
Precisas definir (corretamente!) todas as funções envolvidas!
DEFINIÇÕES.

Sum : List nat → nat Sum [ ] = 0 Sum (x::xs) = x + sum xs	map (α → α) → Lα → Lα map f [ ] = [ ] map f (x::xs) = f x :: map xs

DEMONSTRAÇÃO DE sum (map (·K) ns) = K. sum ns .

Seja ns List nat
Por indução em ns

Base
~~Sum~~ calculamos
Sum (map (·K) [ ]) = sum [ ] [map (·1)]
= 0 [sum (·1)]
calculamos
K. sum [ ] = K·0 [sum (·1)]
= 0 [(·)(·1)]

Passo indutivo
seja xs List nat Tq satisfaz
calculamos
Sum (map (·K) x::xs) = sum (x·K :: map (·K) xs) [map (·2)]
= x·K + sum (map (·K) xs) [sum (·2)]
calculamos
K. sum x::xs = K·(x + sum xs) [pela sum (·2)]
= K·x + K. sum xs [pela dist R]
= Kx + sum (map (·K) xs) [pela HI]
= x.K + sum (map (·K) xs) [pela (·) com]

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

K. Sum

(52) **F**

Para qualquer α : Type, definimos

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

(20) **F1.** Defina as funções:

```
depth  : Tree a → Nat        flatten : Tree a → List a
ntips  : Tree a → Nat        maptree : ?
nforks : Tree a → Nat
```

DEFINIÇÕES.

depth (Tip a) = 0
depth (Fork l r) = S (max (depth l) (depth r))

ntips (Tip a) = S 0
ntips (Fork l r) = ntips l + ntips r

nforks (Tip a) = 0
nforks (Fork l r) = S (nforks l + nforks r)

flatten (Tip a) = [a]
flatten (Fork l r) = flatten l ++ flatten r

maptree : (α → β) → Tree α → Tree β
maptree f (Tip a) = Tip (f a)
maptree f (Fork l r)
= Fork (maptree f l) (maptree f r)

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

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

(32) **F2.** Enuncie e demonstre uma equação interessante sobre *maptree* e a *map*.

DEMONSTRAÇÃO DE flatten (maptree f t) = map f (flatten t).

Por indução (no t)

BASE: flatten (maptree f (Tip a)) = map f (flatten Tip a)

Seja a : α.
Calculamos:

flatten (maptree f (Tip a))
= flatten (Tip (f a))
= [f a]

map f (flatten (Tip a))
= map f [a]
= map f a :: []
!! = ~~(map)~~ (f a) :: map f []
= [f a]

PASSO INDUTIVO:

Sejam l, r : Tree α tais que flatten (maptree f l) = map f (flatten l) & flatten (maptree f r) = map f (flatten r). [H.I.]

calculamos:

flatten (maptree f (Fork l r))
= flatten (Fork (maptree f l) (maptree f r))
= flatten (maptree f l) ++ ~~(maptree)~~ flatten (maptree f r)
= (map f (flatten l)) ++ (map f (flatten r)). [H.I.]

Só isso mesmo.

map f (flatten (Fork l r))
= map f (flatten l ++ flatten r)
= (map f (flatten l)) ++ (map f (flatten r)). [Lema]

Lemmata (até 2)

Lema: $(\forall f : \alpha \to \beta)(\forall xs, ys : \text{List}\,\alpha)[\text{map } f\,(xs +\!\!+\, ys) = (\text{map } f\ xs) +\!\!+\, (\text{map } f\ ys)]$

Sejam $f : \alpha \to \beta$ e $xs, ys : \text{List}\,\alpha$.

Por indução no xs.

BASE: $\text{map } f\,([\,] +\!\!+\, ys) = (\text{map } f\,[\,]) +\!\!+\, (\text{map } f\ ys)$

Calculamos:

$$\begin{aligned}
&\text{map } f\,([\,] +\!\!+\, ys)\\
&\quad = \text{map } f\ ys\\
&(\text{map } f\,[\,]) +\!\!+\, (\text{map } f\ ys)\\
&\quad = [\,] +\!\!+\, (\text{map } f\ ys)\\
&\quad = \text{map } f\ ys
\end{aligned}$$

PASSO INDUTIVO:

Suponha que $\text{map } f\,(xs +\!\!+\, ys) = (\text{map } f\ xs) +\!\!+\, (\text{map } f\ ys)$.

Calculamos:

$$\begin{aligned}
&\text{map } f\,((x :: xs) +\!\!+\, ys)\\
&\quad = \text{map } f\,(x :: (xs +\!\!+\, ys))\\
&\quad = (f x) :: \text{map } f\ xs +\!\!+\, ys\\
&\quad = (f x) :: ((\text{map } f\ xs) +\!\!+\, (\text{map } f\ ys)) \quad [\text{H.I}]\\
&(\text{map } f\ x :: xs) +\!\!+\, (\text{map } f\ ys)\\
&\quad = ((f x) :: \text{map } f\ xs) +\!\!+\, \text{map } f\ ys\\
&\quad = (f x) :: ((\text{map } f\ xs) +\!\!+\, (\text{map } f\ ys)). \quad [\text{outro lema...}]
\end{aligned}$$

(52) **F**

Para qualquer α : Type, definimos

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

$++ : [\alpha] \to [\alpha] \to [\alpha]$

$[\,] ++ ls = ls$

$(u : us) ++ ls = u : (us ++ ls)$

(20) **F1.** Defina as funções:

```
depth   : Tree a → Nat        flatten : Tree a → List a
ntips   : Tree a → Nat        maptree : ?
nforks  : Tree a → Nat
```

DEFINIÇÕES.

ntips (Tip _) = 1
ntips (fork u v) = (ntips u) + (ntips v)

nforks (Tip _) = 0
nforks (Fork u v) = (nforks u) + (nforks v) + 1

max...

depth (Tip _) = 0
depth (Fork u v) = | ~~us > vs = us~~
| ~~otherwise = vs~~
where
us = nfork u
vs nfork v

max us vs

flatten (Tip x) = [x]
flatten (Fork u v) = (flatten u) ++ (flatten v)

map : $(\alpha \to \beta) \to [\alpha] \to [\beta]$
map [ ] = [ ]
map F (x:xs) = Fx : (map f xs)

maptree : $(\alpha \to \beta) \to$ Tree $\alpha \to$ Tree β
maptree g (Tip x) = Tip (g x)
maptree g (Fork u v) = Fork (maptree g u) (maptree g v)

g de gardener pq é uma árvore

(32) **F2.** Enuncie e demonstre uma equação interessante sobre *maptree* e a *map*.

DEMONSTRAÇÃO DE flatten(maptree ^{g} t) = map(^{g} flatten t)

~~Por Indução~~
Seja t : Tree α
Seja g : $\alpha \to \beta$
Por indução:
Seja a : α.

~~Seja t : Tree α~~
~~Seja g : $\alpha \to \beta$~~
Por indução no t.
~~Base : t = Tip α~~
~~flatten (maptree g Tip α) = flatten Tip g(a [maptree.1]~~

Calcule:

flatten (maptree g Tip a) = flatten (Tip (g(a))) [maptree.1]
= [g(a)] [flatten.1]

map g (flatten Tip a) = map g [a] [flatten.1]
= [g(a)] [map.2]

Seja u, v : Tree α

Passo indutivo:

flatten (maptree g (fork u v) = flatten (fork (maptree g u) (maptree g v)) [maptree.2]
= (flatten (maptree g u)) ++ (flatten (maptree g v)) [flatten.2]
= (map g (flatten u)) ++ (map g (flatten v)) [H.I.]

map g (flatten (Fork u v) = map g (flatten u ++ flatten v) [flatten.2]
= (map g (flatten u)) ++ (map g (flatten v)) [(++)-map-distR]

Só isso mesmo.

por que isso?

[(++)-map-distr]

map g (ls ++ us) = (map g ls) ++ (map g us)

Por Indução

map g ([] ++ us) = map g us [++.1]

(map g []) ++ (map g us) = [] ++ map g us [map.1]

= map g us [++.1]

Passo indutivo:

map g ((l:ls) ++ us) = map g (l:(ls ++ us)) [++.2]

= g l : (map g (ls ++ us)) [map.2]

= g l : ((map g ls) ++ (map g us)) [H.I.]

(map g (l:ls)) ++ (map g us) = (g l : (map g ls)) ++ (map g us) [map.2]

(12) **I**

Complete as equações seguintes **com algo interessante:**[3]

```
sum (map (· k) ns) = K · Sum ns
ev (product ns) =
ev (sum ns) =
sorted (map (+ k) ns) = Sorted ns
```

(24) **U**

Escolha **uma** das equações de **I** para demonstrar.
Precisas definir (corretamente!) todas as funções envolvidas!
DEFINIÇÕES.

def Sum : Set ℕ → ℕ
Sum [] = 0
Sum (x :: xs) = x + Sum xs

def map : (Nat → Nat) → Set Nat → Set Nat
Map f [] = []
Map f (x :: xs) = f x :: Map f xs

DEMONSTRAÇÃO DE Sum (map (· K) ns) = K · Sum ns .

Por indução no ns

Base:

Sum (Map (·K) []) = Sum ([])
= 0

K · Sum [] = K · 0
= 0

P.I: Sum (map (·K) x::xs) = Sum (K·x :: map (·K) xs)
o que é isso? → K·x + Sum (map (·K) xs)
K·x + K · Sum xs [HI]
K(x + Sum xs)
K · Sum (x :: xs)

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

(12) **I**

Complete as equações seguintes **com algo interessante:**[3]

✓ sum (map (· k) ns) = ~~(~~sum ns~~)~~ · K
✓ ev (product ns) = anyEven ns
— ev (sum ns) = $\boxed{\text{allEven ns} \parallel}$ ev (length (filter odd ns))
✓ sorted (map (+ k) ns) = sorted ns

por que isso?

(24) **U**

Escolha **uma** das equações de **I** para demonstrar.
Precisas definir (corretamente!) todas as funções envolvidas!
DEFINIÇÕES.

NÃO PRECISOU, PORQUE TAVA DEMONSTRANDO ALGO ERRADO ✓

sum : List Nat → Nat
sum [] = 0
sum (n : ns) = n + sum ns

map : (Nat → Nat) → List Nat → List Nat
map f [] = []
map f (n : ns) = f n : map f ns

ev : Nat → Bool
ev 0 = True
ev S 0 = False
ev S(S n) = ev n

allEven : List Nat → Bool
allEven [] = True
allEven (n : ns) = ev n && allEven ns

DEMONSTRAÇÃO DE ~~ev (sum ns) = allEven ns~~ sum (map (·K) ns) = K · sum ns

~~Seja l : List Nat.~~
~~Por indução em l:~~
~~Caso []:~~
~~Calculamos:~~
~~ev (sum []) = ev (0) [sum.1]~~
~~= True [ev.1]~~
~~allEven [] = True [allEven.1]~~
~~Caso (n : ns)~~
~~Alvo:~~
~~ev (sum ns) = allEven ns ⇒~~
~~ev (sum n : ns) ⇒ allEven ns~~

~~Calculamos:~~
~~ev (sum n : ns) = ev (n + sum ns)~~
~~= ev n && ev (sum ns)~~
~~= ev n && allEven ns [hip. indut.]~~
~~= allEven ns [allEven~~

~~TÁ ERRADO~~

VOU DEMONSTRAR sum (map (·K) ns) = K · sum ns
Por indução em l : List Nat.
Caso []
sum (map (·K) []) = sum []
= 0
K · sum [] = K · 0
= 0

Caso (n : ns) Calculamos: sum (map (·K) (n : ns)) = sum (n · K + map (·K) ns) [map.2]
= sum (map (·K) ns) + (n · K)
= (sum ns) · K + n · K [hip. indutiva]
= K · (n + sum ns) = K · (sum (n : ns))

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

(52) **F**

Para qualquer α : Type, definimos

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

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

(20) **F1.** Defina as funções:

```
depth   : Tree a → Nat
ntips   : Tree a → Nat
nforks  : Tree a → Nat
flatten : Tree a → List a
maptree : ?
```

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

DEFINIÇÕES.

depth (Tip a) = 0
depth (t 'fork' t') = 1 + max(depth t, depth t')
ntips (tip a) = 1
ntips (t 'fork' t') = ntips t + ntips t'
nforks (Tipa) = 0
nforks (t 'fork' t') = 1 + nforks t + nforks t'
flatten (tip a) = [a]
flatten (t 'fork' t') = flatten t ++ flatten t'

mapTree : Tree α -> (α -> β) -> Tree β
maptree f (Tip α) = Tip (f α)
maptree f (t 'fork' t) = (maptree f t) 'fork' (mapTree f t')

(32) **F2.** Enuncie e demonstre uma equação interessante sobre *maptree* e a *map*.

DEMONSTRAÇÃO DE map f (flatten t) = flatten (maptree f t)

map f (flatten t) = flatten (maptree f t)

Por indução em t, separe em 2 casos.

caso t = tip a.
map f (flatten (tip a)) =
[flatten.1] map f [a] = [f a]

flatten (mapTree f tip a) = [maptree.1]
flatten (Tip fa) = [fa] [flatten.1]

caso t = t_1 'fork' t_2
map f (flatten (t_1 'fork' t_2))
[flatten 2] map f (flatten t_1 ++ flatten t_2)
[demonstra] (map f (flatten t_1)) ++ (map f (flatten t_2))

flatten (maptree f (t_1 'fork' t_2)) [maptree.2]
flatten (maptree f t_1 'fork' maptree f t_2) [flatten.2]
flatten (maptree f t_1) ++ flatten (maptree f t_2)

hipótese indutiva

Só isso mesmo.

parece rascunho
nenhum verbo (=) em todo isso?!

ao que isso refere??

LEMMATA (até 2)

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

indução em xs

caso [ ]

map f ([ ] ++ ys)

= map f ys

= ~~map f y : map f ys~~

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

[ ] ++ (map f ys)

map f ys

caso (x:xs)

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

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

f x : map f (xs ++ ys)

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

???

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

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

hipótese indutiva

nenhum verbo!?

(52) **F**

Para qualquer α : Type, definimos

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

(20) **F1.** Defina as funções:

```
depth   : Tree a → Nat          flatten : Tree a → List a
ntips   : Tree a → Nat          maptree : ?
nforks  : Tree a → Nat
```

DEFINIÇÕES.

depth (T_) = 0
depth (F l r) = S(max(depth l)(depth r))

ntips (T_) = S0
ntips (F l r) = ntips l + ntips r

nforks (T_) = 0
nforks (F l r) = S(nforks l + nforks r)

flatten (T x) = [x]
flatten (F l r) = flatten l ++ flatten r

??

mtree : Tree α → (α → α → α) → α → Tree α
mtree (T x) f n = T (f x n)
mtree (F l r) f n = F(mtree l f n)(mtree r f n)

(32) **F2.** Enuncie e demonstre uma equação interessante sobre *maptree* e a *map*.

DEMONSTRAÇÃO DE map l f n = flatten (mtree t f n)

X

Por indução no l.

Caso []: map [] f n = [] [map.1]

?? = flatten (mtree t f n).

Caso (x:xs): map (x:xs) f n = f n x : map xs f n [map.2]
= f n x : flatten (mtree t f x) [H1]
= f n x : xs
= (x:xs)
= flatten (mtree t f n).

X

Só isso mesmo.

(52) **F**

Para qualquer α : Type, definimos

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

(20) **F1.** Defina as funções:

```
✓depth  : Tree a → Nat        ✓flatten : Tree a → List a
✓ntips  : Tree a → Nat        ✓maptree : ?
✓nforks : Tree a → Nat
```

DEFINIÇÕES.

depth (T_) = 0
depth (F l r) = S (max (depth l)(depth r))

nTips (T_) = 1
nTips (F l r) = nTips l + nTips r

nfork (T_) = 0
nfork (F l r) = S (nfork l + nfork R)

FLATTEN (T x) = [x]
FLATTEN (F l r) = FLATTEN l ++ FLATTEN r

MAPTREE : ($\alpha \to \beta$) → TREE α → TREE β
MAPTREE F (T x) = T (F x)
MAPTREE F (FORK L R) ~~= MAPTREE F L~~
= FORK (MAPTREE F L)(MAPTREE F R)

MAP : ($\alpha \to \beta$) → List α → List β
MAP F [] = []
MAP F (x:xs) = F x : MAP F xs

(32) **F2.** Enuncie e demonstre uma equação interessante sobre *maptree* e a *map*.
DEMONSTRAÇÃO DE ~~MAPTREE F T = MAP F (FLATTEN T)~~.

FLATTEN (MAPTREE F T) = MAP F (FLATTEN T)

Por indução em T

caso Tx:

FLATTEN T (Fx) [MAPTREE.1]
= [Fx] [FLATTEN.1]

sem (=)!

MAP F [x] [FLATTEN.1]
Fx : MAP F [] [MAP.2]
Fx : [] [MAP.1]
[Fx]

CASO ~~FER~~ FORK L R:

Só isso mesmo.

(12) **I**

Complete as equações seguintes **com algo interessante:**[3]

```
sum (map (· k) ns) ≠ sum ns + k · len ns
ev (product ns) =
ev (sum ns) ≠ odd (sum (ns :: [1]))
sorted (map (+ k) ns) = sorted ns .
```

(24) **U**

Escolha **uma** das equações de **I** para demonstrar.
Precisas definir (corretamente!) todas as funções envolvidas!
DEFINIÇÕES.

map : (α → β) → List α → List β map f (a :: as) = f a :: map f as map f _ = []	sorted : List α → bool sorted (x :: x' :: xs) | x ≤ x' = True && sorted (x' :: xs) | otherwise = false. sorted _ = True

substitua por algo também

DEMONSTRAÇÃO DE sorted (map (+k) ns) = sorted ns .

Por indução:
caso []:
calc:
sorted (map (+k) [] = sorted []. [map.2]

devido a def, tu precisa tratar o caso [x] separadamente

Passo indutivo: — faltou o padrão
Suponha sorted (map (+k) (n' :: ns)) = sorted (n' :: ns)
logo, caso n ≤ n': — n + k ≤ n' + k
quem é?
calc:
[map.1] → sorted (map (+k) n :: n' :: ns) = sorted (n + k :: n' + k :: map (+k) ns)
[sorted.1] = True && sorted (n' + k :: map (+k) ns)
[← map.1] = True && sorted (map (+k) n' :: ns)
[H.I.] = True && sorted (n' :: ns)
[sorted.1] sorted (n :: n' :: ns) = True && sorted (n' :: ns).
caso n > n': similar.

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

(12) **I**

Complete as equações seguintes **com algo interessante:**[3]

```
sum (map (· k) ns) = [(x1·k)+(x2·k)+(x3·k)...+(xn·k)]
ev (product ns) = Ev[x1·x2·x3... xn] = True – ELSE FALSE ??
ev (sum ns) = Ev[x1+x2+x3 ... xn] = True
sorted (map (+ k) ns) = [(x1+k)>(x2+k)>(x3+k)...>(xn+k)] = True / ELSE => FALSE
```

(24) **U**

Escolha **uma** das equações de **I** para demonstrar.
Precisas definir (corretamente!) todas as funções envolvidas!
DEFINIÇÕES.

sum(map (·k)ns) = LN => Nat
sum(map (·k) [ ] = Erro
sum(map(·k) [Ln] = (sum(LN·k))

DEMONSTRAÇÃO DE ~~sum~~ [(x1 · k) + (x2 · k) + (x3 · k) ... + (xn · k)]

?

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