(Veja o PDF IntroProof.)

lec01 – lec06
2025-08-18 – 2025-09-03

Os Nats

lec07
2025-09-10

- o O é um Nat
- se n é um Nat, então S n é um Nat
- nada mais (é um Nat)

(meta) variável

Uns Nats: O, S O, S (S O), ..

... usando regras

$$\frac{}{0 : \mathsf{Nat}}\ \textsc{Zero} \qquad \frac{n : \mathsf{Nat}}{S\ n : \mathsf{Nat}}\ \textsc{Succ}$$

Inferimos:

$$0 : \mathsf{Nat}, \qquad S : \mathsf{Nat} \to \mathsf{Nat}$$

$$\dfrac{\dfrac{\dfrac{}{0 : \mathsf{Nat}}\ \textsc{Zero}}{S\,0 : \mathsf{Nat}}\ \textsc{Succ}}{S\,(S\,0) : \mathsf{Nat}}\ \textsc{Succ}$$

.. listando os construtores

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

.. listando as formas

data Nat ≡ 0 | S Nat

↑ escondida disjunção

data Nat ≡ 0 | S Nat

$$(\forall n : \mathrm{Nat})[\, n = 0 \;\vee\; (\exists n')[\, n = \mathrm{S}\, n' \,]\,]$$

n é o 0

n é succ de algum Nat

Açúcar sintáctico

0	Sug	0	Sug	0
1	Sug	S0	Sug	S 0
2	Sug	SS0	Sug	S (S 0)
3	Sug	SSS0	Sug	S(S(S 0))
⋮	⋮	⋮	⋮	⋮

$(+)$ ou $\_+\_$

$(+) : \mathbb{N}at \times \mathbb{N}at \to \mathbb{N}at$

$n + 0 = n$

$n + Sm = S(n+m)$

$(+) : \mathbb{N}at \to (\mathbb{N}at \to \mathbb{N}at)$

$n + m \equiv$

$0 + 0 = 0$

$0 + Sm = Sm$

$Sn + 0 = Sn$

$Sn + Sm = S(S(n+m))$

$\Theta.\ (\forall m)[\, Sm = m + 1 \,]$ $\equiv$ SO

$\Theta.\ \ 2 + 3 = 5$

$\alpha \to \alpha \to \quad \to \alpha$

η

Calculando

$$
\begin{aligned}
(+) &: \mathrm{Nat} \times \mathrm{Nat} \to \mathrm{Nat} \\
n + 0 &= n \\
n + Sm &= S(n + m)
\end{aligned}
$$

$$
\begin{aligned}
& 2 + 3 \\
&\equiv \overbrace{SS0}^{n} + \overbrace{SSSS0}^{m} \\
&= S(\overbrace{SS0}^{n} + S\overbrace{SS0}^{m}) && [(+).2 \text{ com } n := SS0,\ m := SS0] \\
&= S(S(\overbrace{SS0}^{n} + S\overbrace{S0}^{m})) && [(+).2 \ n := SS0,\ m := S0] \\
&= S(S(S(\overbrace{SS0}^{n} + 0))) && [(+).2 \ n := SS0,\ m := 0] \\
&= S(S(S(SS0))) && [(+).1 \ n := SS0] \\
&\equiv SSSSS0 \\
&\equiv 5
\end{aligned}
$$

Θ. 0-idR-(+) $(\forall n)[\, n + 0 = n \,]$

Seja n : Nat.

Calc: $n + 0 = n$. [(+).1 com n := n]

Θ. 0-idL-(+) $(\forall n)[\, 0 + n = n \,]$

Seja n . Nat.

Calc:

$0 + n$

= não tem como casar o padrão $n + Sm$ com a expressão $0 + n$.

(+) : Nat × Nat → Nat

$n + 0 = n$

$n + Sm = S(n + m)$

-- Vamos tentar separar em casos a partir da forma do n:

por que eu escrevi assim?

Caso n =: 0 :

Calc:

0 + 0

= 0 . [(+).1 n := 0]

Caso n =: S n'

Calc:

0 + S n'

= S (0 + n') [(+).2 n := 0, m := n']

= travamos novamente!

Agora precisamos saber a forma de n'.

Suspeita: não dá.

não virou teorema

A proposição O-idL-(+) sobreviveu!

Número vs numeral

lec08
2025-09-12

→ numeráis representando o mesmo número

α	β	γ	δ	...
I	II	III	IV	...
1	2	3	4	...
1	10	11	100	...
um	dois	três	quatro	...

Seja b a quantidade de brasileiros no mundo.

Quantos dígitos tem o b?

↑ Não faz sentido! (Type Error)

Números não têm dígitos!

Disjunção escondida

data Nat ≡ O | S Nat

↑ escondida disjunção:

$$(\forall n : Nat)\left[\ \underbrace{n = O}_{n \text{ é o } O} \ \vee \ \underbrace{(\exists n')[\, n = S\ n' \,]}_{n \text{ é sucesor de algum Nat}}\ \right]$$

(=) (∃)

6. (+)-assoc $\iff (\forall x, y, z)\left[\underbrace{(x + y) + z}_{\equiv (x+y)+z} = x + (y + z)\right]$

(+) ou $-+-$

$(+) : Nat \times Nat \to Nat$

$n + 0 = n$

$n + Sm = S(n + m)$

+ assoc-L

Sintaxe vs semântica

este símbolo denota uma operação

$$\_ \heartsuit \_ \;:\; \alpha \times \alpha \longrightarrow \alpha$$

$x \heartsuit y \heartsuit z \overset{\text{Sug}}{=\!=} (x \heartsuit y) \heartsuit z$

$x \heartsuit y \heartsuit z \overset{\text{Sug}}{=\!=} x \heartsuit (y \heartsuit z)$

$(\heartsuit)$ é associativa : Prop

$\Updownarrow$

$(\forall x, y, z : \alpha)[\, (x \heartsuit y) \heartsuit z = x \heartsuit (y \heartsuit z) \,]$

atribua associatividade-L ao $\heartsuit$: Cmd

$\rightsquigarrow \quad x \heartsuit y \heartsuit z \overset{\text{Sug}}{=\!=} (x \heartsuit y) \heartsuit z$

atribua associatividade-R ao $\heartsuit$: Cmd

$\rightsquigarrow \quad x \heartsuit y \heartsuit z \overset{\text{Sug}}{=\!=} x \heartsuit (y \heartsuit z)$

6. (+)-assoc $\iff (\forall x, y, z)\left[(x + y) + z = x + (y + z) \right]$

Sejam $a, b, c : \mathbb{N}at$.

Sep em casos. $[\, c : \mathbb{N}at \,]$

$a : \mathbb{N}at \quad (a + b) + c = a + (b + c)$

$b : \mathbb{N}at$

$c : \mathbb{N}at$

$(+) : \mathbb{N}at \times \mathbb{N}at \to \mathbb{N}at$

$n + 0 = n$

$n + Sm = S(n + m)$

Caso $c = 0$:

Calc:

$(a + b) + \underline{c}$

$= \underline{(a + b) + 0}$ [hip. (do caso)]

$= a + b$ [(+).1 $n := a + b$]

$a + (b + \underline{c})$

$= a + \underline{(b + 0)}$ [hip.]

$= a + b$ [(+).1 $n := b$]

CASO $(\exists c')[\, c = S\,c' \,]$:

Seja c' t.q. $c = S\,c'$. [hip.] ou: Logo seja c' t.q. ...

(sem justificativa graças ao «Logo...»)

Calc:

$$(a+b)+c$$
$$= (a+b)+S\,c' \qquad [\text{pela (+) escolha de } c'] \quad \text{(maneira ótima de referir a esse dado sem precisar de rótulo!)}$$
$$= S((a+b)+c') \qquad [\,(+).2 \;\; n := a+b,\; m := c'\,]$$

$$a+(b+c)$$
$$= a+(b+S\,c')$$
$$= a+S(b+c')$$
$$= S\,(a+(b+c'))$$

$(+) : \mathrm{Nat} \times \mathrm{Nat} \to \mathrm{Nat}$

$n + 0 = n$

$n + S\,m = S\,(n+m)$

-- ... , $S((a+b)+c) = S(a+(b+c))$ ⊢ ...

Basta: $S((a+b)+c') = S(a+(b+c'))$ [?]

$$
\begin{array}{ccc}
\| & & \| \\
(a+b)+c & \overset{?}{=} & a+(b+c)
\end{array}
$$

Basta: $(a+b)+c' = a+(b+c')$ [?]

⚠ Qualquer uso de «Basta: _» precisa ser justificado.

E desistimos:

continuando assim não tem como fechar esta demonstração.

Basta

DEMONS:

⋮

Basta demonstrar: R .

isso é uso de (⇒).

Primeiro contato com Lean. (Veja gravação.)

lec09

2025-09-15

Primeiro contato com Haskell. (Veja gravação.)

lec10

2025-09-17

Presente da recursão

lec11
2025-09-22

o que tá sendo definido implicitamente

```
fact : Nat → Nat
fact 0     ≡ S0
fact (S n) ≡ fact n · S n
```

o que tá sendo definido explicitamente

presente da recursão

Sobre as parens implicitas:

$(\mathit{fact}\ n) \cdot (S\ n)$

Indução como regra (uma?) de inferência

os construtores preservam a φ

os construtores preservam a φ

0 é confiável — BASE

S é confiável — PASSO INDUTIVO

$$\frac{\varphi(0) \qquad (\forall k : Nat)[\ \varphi(k) \Rightarrow \varphi(S\,k)\ ]}{(\forall n : Nat)[\ \varphi(n)\ ]}\ IND_{\varphi}$$

$\varphi : Nat \to Prop$

Comando

efeito instantâneo; compare com split/sep.:

Por indução!

BASE: -- $\cdots \vdash \varphi(0)$

PASSO INDUTIVO: -- $\cdots \vdash (\forall k)[\varphi(k) \Rightarrow \varphi(Sk)]$

Seja k t.q. $\varphi(k)$. -- ALVO: $\varphi(Sk)$

(ou)

PASSO INDUTIVO: -- $\cdots \vdash (\forall k)[\overbrace{\varphi(k) \Rightarrow \varphi(Sk)}^{\psi(k)}]$

Por indução! $\psi(0)$

BASE: -- $\varphi(0) \Rightarrow \varphi(Sk)$

P.I.: -- $(\forall k)[(\underbrace{\varphi(k) \Rightarrow \varphi(Sk)}_{\psi(k)}) \Rightarrow (\underbrace{\varphi(Sk) \Rightarrow \varphi(SSk)}_{\psi(k)})]$

Split.
PARTE L:
PARTE R:

Sep em casos..
CASO L:
CASO R:

6. (+)-assoc $\iff (\forall x, y, z)\left[(x + y) + z = x + (y + z) \right]$

Seja a : Nat . $(\forall y)(\forall z)\left[(a + y) + z = \cdots \right.$

Seja b : Nat . $(\forall z)\left[(a + b) + z = \cdots \right.$

Seja c : Nat . $(a + b) + c = \cdots$

Por indução!

BASE $((a + b) + 0 = a + (b + 0))$:

(Já feito.)

P.I. $((\forall k)[(a+b) + k = a + (b+k)$

$\Rightarrow (a+b) + S\,k = a + (b + S\,k)]$ ← H.I.

Seja k t.q. $(a + b) + k = a + (b + k)$.

↑ um k legalzinho: associa com o a e o b assim

Calc:

$$(a + b) + S\,k$$

$$= S\,((a+b) + k) \qquad [\,(+).2 \qquad n := a+b,\ m := k\,]$$

$$= S\,(a + (b + k)) \qquad [\,H.I.\,]$$

$$a + (b + S\,k)$$

$$= a + (S\,(b + k)) \qquad [\,(+).2 \cdots\,]$$

$$= S\,(a + (b + k)) \qquad [\,(+).2 \cdots\,]$$

Indução mais cedo ...

Θ. (+)-assoc $\iff (\forall x, y, z)\left[(x + y) + z = x + (y + z) \right]$

Seja a : Nat .

$(\forall y)\underbrace{(\forall z)\left[(a + y) + z = \cdots \right.}_{\varphi(y)}$

Por indução!

BASE: $(\forall z)\left[(a + 0) + z = a + (0 + z) \right]$

P.I. : ?

...e mais cedo ainda

0. (+)-assoc $\iff (\forall x,y,z)\left[(x+y)+z = x+(y+z) \right]$

Basta demons: $(\forall z)\underbrace{(\forall x,y)\left[(x+y)+z = x+(y+z) \right]}_{\varphi(z)}$

Por indução. ↳ justifique!

BASE: $(\forall x,y)\left[(x+y)+0 = x+(y+0) \right]$

⋮

P.I.: $(\forall k)\left[\begin{array}{l} (\forall x,y)\left[(x+y)+k = x+(y+k) \right] \\ \Rightarrow (\forall x,y)\left[(x+y)+Sk = x+(y+Sk) \right] \end{array} \right]$

Seja k t.q. $(\forall x,y)\left[(x+y)+k = x+(y+k) \right]$

↑ legalzão: ele associa com quaisquer nats assim $(\_+\_)+k$

⋮

Indução como match com presente

DADOS ALVO

n : Nat ...

Por indução no n : Nat.

meu Nat — um padrão

CASO n =: O :

não é uma equação, mas sim um pattern-matching

CASO n =. S n :

meu Nat — um padrão

nestes momentos eu tenho mesmo escolha sobre qual Nat botarei aqui para casar com tal padrão?

Recursão ? vs Indução

« Definir por recursão; demonstrar por indução »

Θ. $(\forall n : Nat)\ [\ \varphi(n)\ ]$

Vou definir uma $f : Nat \to Proof$?

que recebendo qualquer Nat x, constroi e retorna uma demonstração de $\varphi(x)$.

Por recursão:

$f\ 0 \equiv$ [demons de $\varphi(0)$]

$f\ (S\ n) \equiv$ [demons de $\varphi(S\ n)$]

↑ aqui tenho acesso à $(f\ n)$, que é uma demons de $\varphi(n)$.

} Presente da recursão. Isto é a hipótese indutiva!

Não abusarás ...

lec12
2025-09-24

abuso de Nats!

```
even  :  Nat → Nat
_≤_   :  ?
_<_   :  ?
```

... Defina um novo tipo!

```
data Bool = ff | tt
```

```
data Bool
  ff : Bool
  tt : Bool
```

```
if_then_else_ : Bool → α → α → α
ou (Bool × α × α) → α
```

Weekday

lec13
2025-09-26

```
data  Weekday
        Mon  : Weekday
        Tue  : Weekday
          ⋮        ⋮
        Sun  : Weekday
```

```
data  Weekday = Mo | Tu | We | Th | Fr | Sa | Su
```

Unit ("void")

```
data Unit
    * : Unit
```

tendo apenas um habitante, tal habitante não carrega nenhuma informação.
Mesmo assim é necessário como "moedinha" para usar funções nulárias.

quem é? quantos/quais habitantes tem?

```
int f (void)
    return 42;
```

essa seria inchamável!

$f : \emptyset \to X$

C: `f()` `f(32, 12, 3)`

nós: `f␣()` `f␣(32, 12, 3)`

a 0-tupla

Empty

```
data Empty
```

nenhum construtor
portanto: nenhum habitante

HW: procure funções saindo de e funções entrando em ... cada um desses tipos novos.

Aridade, (des)currificação, aplicação parcial

$\hat{f} : (\alpha \times \beta \times \gamma) \to \delta$ ← 3-ária descurrificada

$f : \alpha \to (\beta \to (\gamma \to \delta))$ ← 3-ária currificada

$\hat{g} : \text{Unit} \to \delta$ ← 0-ária descurrificada

$g : \delta$ ← 0-ária currificada

```
def f(x):
    def g(y):
        return x + y
    return g
```

$(f\ (42))\ (7) = ?$

```
f x y = x + y
f 42 7 = ...
```

HW: Escreva, teste, execute, e entenda isso em Python.

ListNat

```
data ListNat
  Nil  : LN
  Cons : N → LN → LN
```

head — tail

```
[0,1,0]
[42]
[1,1,1,1]
[]
[1,1,1,...]
[0,1,2,...]
```

Açúcares:

`[]` sug≡ `Nil`

`(::)` sug≡ `Cons`

$1 :: (2 :: (3 :: []))$

$[a_1, a_2, \ldots, a_n]$ sug≡ $a_1 :: a_2 :: \cdots :: a_n :: []$

Cadê a recursão?

```
data Nat = O | S Nat
```

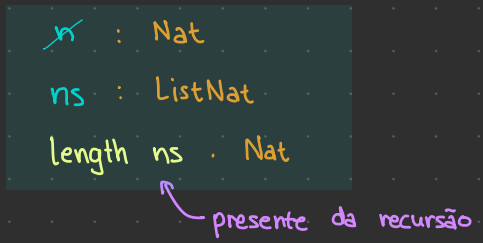
```
data Nat
  O : Nat
  S : Nat → Nat
```

$$\frac{}{O : Nat} \qquad \frac{n : Nat}{S\ n : Nat}$$

Recursão em listas

```
length : ListNat → Nat

length []       = 0
length (n :: ns) = S (length ns)
```



Calculando...

```
length [42, 3, 3, 5]

≡ length (42 :: [3, 3, 5])

= S (length [3,3,5])      [length.2 _ := 42  ns := [3,3,5]]
= S (S (length [3,5]))    [length.2 ...]
⋮
= S (S (S (S (length []))))
= SSSS0
```

De ListNat para List α para List Nat

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

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

```
head : ?
?
```

```
head [42, 3, 3, 12, 0] = 42
```

```
get : ?
?
```

```
get 3 [42, 3, 3, 12, 0] = 12
      0   1  2  3
```

Habitantes de List α

lec14
2025-09-29

List Unit:

`[]`	`Nil`
`[*]`	`Cons * []`
`[*, *]`	`Cons * it`
`[*, *, *]`	`Cons * it`
⋮	

List Empty:

`[]`	`Nil`

(acabou)

Desconstrutores de construtores

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

```
pred : Nat → Nat        (função parcial)
pred (S n) = n
```

```
data List α       List : Type → Type
  Nil : List α
  Cons : α → List α → List α
```

```
hd : List α → α          tl : List α → List α
hd [3,0,5] = 3           tl [3,0,5] = [0,5]
```

Concatenação (++)

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

$$[] \mathbin{++} ys = ys$$

$$(x :: xs) \mathbin{++} ys = x :: (xs \mathbin{++} ys)$$

$$[] \mathbin{++} [] = []$$

$$xs \mathbin{++} [] = xs$$

$$[3,4,5] \mathbin{++} []$$

$$= 3 :: ([4,5] \mathbin{++} [])$$

$$= 3 :: (4 :: ([5] \mathbin{++} []))$$

$$= 3 :: 4 :: (5 :: ([] \mathbin{++} []))$$

$$= 3 :: 4 :: 5 :: []$$

[(++).2 x := 3, xs := [4,5], ys := []]

$$[2,0,3] \mathbin{++} [2,7,9,8] = [2,0,3,2,7,9,8]$$

$x : \alpha$

$y : \alpha$

$xs : [\alpha]$

$ys : [\alpha]$

$xs \mathbin{++} ys : [\alpha]$

$xs \mathbin{++} (y :: ys)$

$(x :: xs) \mathbin{++} ys$

descrever

x := 2

y := 2

xs := [0,3]

ys := [7,9,8]

xs ++ ys := [0,3,7,9,8]

xs ++ (y::ys) := [0,3,2,7,9,8]

(x::xs) ++ ys := [2,0,3,7,9,8]

Mais funções

reverse : L α → L α

rev [3,0,0,7] = [7, 0, 0, 3]

sort : L Nat ← ou qualquer tipo ordenável

sort [3,0,0,7] = [0, 0, 3, 7]

$n \leq m \overset{\text{def}}{\iff} (\exists k)[\, n + k = m \,]$

$n < m \overset{\text{def}}{\iff} (\exists k)[\, n + S\, k = m \,]$

HW. $(\leq) \rightleftarrows (<)$

$(<)$-irrefl $\iff a \not< a$

$(<)$-asym $\iff a < b \Rightarrow b \not< a$

Internalização de conceito

$(\leq) : \mathrm{Nat} \times \mathrm{Nat} \to \mathrm{Prop}$

↓

$\mathrm{leq} : \mathrm{Nat} \xrightarrow{\times} \mathrm{Nat} \to \mathrm{Bool}$

⊖. $\mathrm{leq}\ n\ m = \mathrm{True} \iff n \leq m$

Procurando teoremas

- $\ominus$. $\text{len}\ (xs +\!\!+ ys) = \text{len}\ xs + \text{len}\ ys$
- $\ominus$. $\text{rev}\ (\text{rev}\ xs) = xs$
- $\ominus$. $(\forall xs : L\,\alpha)\,[\text{len}\ (\text{rev}\ xs) = \text{len}\ xs]$
- $\ominus$. $(+\!\!+)$-assoc. $(xs +\!\!+ ys) +\!\!+ zs = xs +\!\!+ (ys +\!\!+ zs)$

Indução (List α)

$$\frac{\varphi(\text{Nil}) \qquad (\forall x : \alpha)(\forall l : L\,\alpha)[\,\varphi(l) \Rightarrow \varphi(x :: l)\,]}{(\forall l . L\,\alpha)[\,\varphi(l)\,]}\ IND_{\varphi}^{\text{List } \alpha}$$

Nil preserve φ

Cons preserve φ

Exemplo de indução

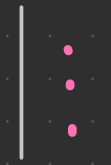
Θ. $\text{len}\ (xs +\!\!+ ys) = \text{len}\ xs + \text{len}\ ys$

Indução no xs.

CASO `[]` : -- ALVO: $\text{len}\ ([] +\!\!+ ys) = \text{len}\ [] + \text{len}\ ys$

CASO $(x :: xs')$: -- $\text{len}\ ((x :: xs') +\!\!+ ys) = \text{len}\ (x :: xs') + \text{len}\ ys$

-- H.I: $\text{len}\ (xs' +\!\!+ ys) = \text{len}\ xs' + \text{len}\ ys$



Indução de um tipo indutivo α

lec15
2025-10-01

$$\frac{\text{os construtores preservam a } \varphi}{(\forall x : \alpha)[\ \varphi(x)\ ]}\ \text{IND}_\alpha^\varphi \qquad \varphi : \alpha \to \text{Prop}$$

Exemplo: considere um tipo B definido assim:

data B

C_1 : B

C_2 : B → Nat → B → String → B

C_3 : B → B → B → B

C_4 : Nat → Nat → B

Como apareceria uma demons de $(\forall b : B)[\, \varphi(b)\,]$:

Seja $b : B$. -- $b . B \vdash \varphi(b)$

Indução no $b : B$.

CASO C_1: -- ALVO: $\varphi(C_1)$

| DADOS . — + HIs : —

CASO $C_2\ b'\ n\ b''\ s$: -- ALVO . $\varphi(C_2\ b'\ n\ b''\ s)$

| DADOS: $b' : B$, n . Nat, $b'' : B$, s : String + HIs : $\varphi(b')$, $\varphi(b'')$

CASO $C_3\ b_1\ b_2\ b_3$: -- ALVO · $\varphi(C_3\ b_1\ b_2\ b_3)$

| DADOS : $b_1, b_2, b_3 : B$ + HIs : $\varphi(b_1)$, $\varphi(b_2)$, $\varphi(b_3)$

CASO $C_4\ n\ m$: -- ALVO: $\varphi(C_4\ n\ m)$

| DADOS : n, m : Nat + HIs : —

|

Exemplo: indução de Bool

$\vdash (\forall b : \mathrm{Bool})\,[\ \varphi(b)\ ]$

Seja b. Bool

Indução no b.

Caso ff:

| -- Alvo: $\varphi(\mathit{ff})$

Caso tt.

| -- Alvo: $\varphi(\mathit{tt})$

Princípios sobre igualdade de tipos indutivos

Disjointness of constructors:

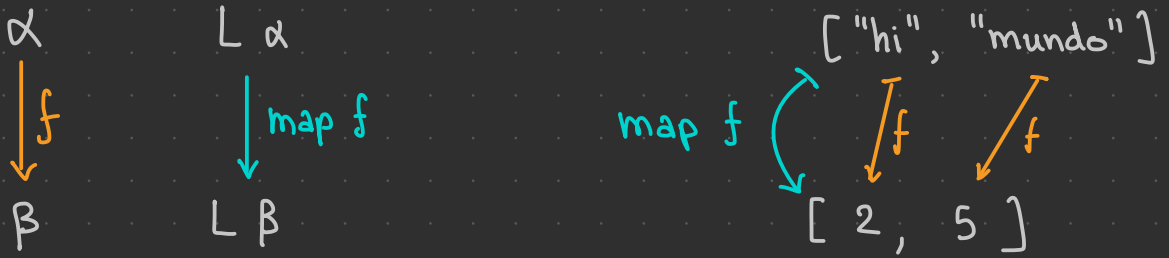
Construtores distintos constroem valores distintos

Injectivity of constructors:

O mesmo construtor aplicado em valores distintos constrói valores distintos.

« deriving Eq »

map



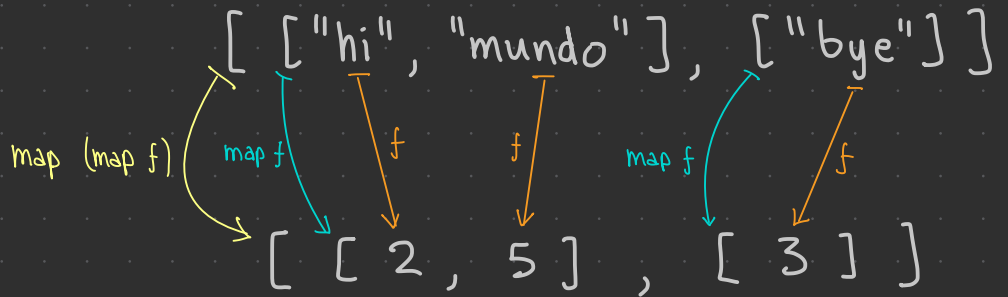

```
map : (α → β) → L α → L β
map f []     = ...
map f (x:.xs) = ...
```

visualizado como binária:
recebendo uma função e uma lista
retorna uma nova lista.

```
map : (α → β) → (L α → L β)
```

visualizado como unária: transforma funçãozinhas para funçãozonas

etc. . .



filter

filter : $(\alpha \to \text{Bool}) \to L\,\alpha \to L\,\alpha$

: