lec09

2024-11-11

Nat : Type

S : Nat → Nat

O : Nat

Bool : Type

False : Bool

True : Bool

Ints		Nats
0	≡ MI	0
1	≡ MI	2
2	≡ MI	4
3	≡ MI	6
⋮		
−1	≡ MI	1
−2	≡ MI	3
−3	≡ MI	5
⋮		

Ints

Nats

⚠ $\mathbb{N} \not\subseteq \mathbb{Z}$

Int : Type ?

MkInt : Nat → Int

Neg : Int → Int

ou

MkInt : Nat → Int

wrapper

ou

MkInt : Nat → Nat → Int

que tal Sign?

que tal Bool?

data Sign
- P : Sign
- Z : Sign
- N : Sign

ou

Z : Int

SP : Int → Int

SN : Int → Int

3 ≡ SP (SP (SP Z))

−2 ≡ SN (SN Z)

0 ≡ Z

SN (SP (SP Z))

especificação de inteiros

$(\mathbb{Z}\,;\, +\,,\, \cdot\,,\, -\,,\, 0\,,\, 1\,,\, Pos\,)$ + axiomas

$\mathbb{Z}_{\mathbb{N}} \overset{\text{def}}{=} \{0_{\mathbb{Z}}, 1_{\mathbb{Z}}, 2_{\mathbb{Z}}, \dots\} = \{0\} \cup Pos$

Polimorfismo

$\mathrm{idNat} : \mathrm{Nat} \to \mathrm{Nat}$

$\mathrm{idNat}\ n = n$

$\mathrm{idBool} : \mathrm{Bool} \to \mathrm{Bool}$

$\mathrm{idBool}\ b = b$

$\boxed{\begin{array}{l} \mathrm{id}_{\alpha : \mathrm{Type}} : \alpha \to \alpha \\ \mathrm{id}\ x = x \end{array}}$

isso define uma
α-indexada família de funções

("open-ended")

$\mathrm{ListNat} : \mathrm{Type}$

$\mathrm{Cons} : \mathrm{Nat} \to \mathrm{ListNat} \to \mathrm{ListNat}$

$\mathrm{Nil} : \mathrm{ListNat}$

$$\frac{\mathrm{List} : \mathrm{Ty} \to \mathrm{Ty} \qquad \alpha : \mathrm{Ty}}{\mathrm{List}\ \alpha : \mathrm{Ty}}$$

data $\mathrm{List}\ \alpha$

$\mathrm{Nil} : \mathrm{List}\ \alpha$

$\mathrm{Cons} : \alpha \to \mathrm{List}\ \alpha \to \mathrm{List}\ \alpha$

$\mathrm{length} : \mathrm{List}\ \alpha \to \mathrm{Nat}$

$\mathrm{sum} : \mathrm{List}\ \mathrm{Nat} \to \mathrm{Nat}$

Recursão em listas

lec10
2024-11-25

$$(+\!\!+) : L\,\alpha \to L\,\alpha \to L\,\alpha$$

$$xs +\!\!+ [\,] = xs$$

$$xs +\!\!+ (y :: ys) = \underbrace{\dots\dots\dots\dots}_{\text{eu sei } xs +\!\!+ ys}$$

$$[\,] +\!\!+ ys = ys$$

$$(x :: xs) +\!\!+ ys = \underbrace{x :: (xs +\!\!+ ys)}_{\text{eu sei } xs +\!\!+ ys}$$

$$[1, 2, 3] +\!\!+ [5, 6]$$
$$= 1 :: ([2,3] +\!\!+ [5,6])$$
$$= 1 :: (2 :: ([3] +\!\!+ [5,6]))$$
$$= 1 :: (2 :: (3 :: ([\,] +\!\!+ [5,6])))$$
$$= 1 :: (2 :: (3 :: [5,6]))$$
$$\equiv [1, 2, 3, 5, 6]$$

$$[1, 2, 3] +\!\!+ [8, 7, 9]$$

$$[1, 2, 3] +\!\!+ [7, 9]$$

$$[1, 2, 3, 7, 9] \xrightarrow{\text{insertAt (length xs) 8}} [1, 2, 3, 8, 7, 9]$$

complicado

simples

$$[2, 3] +\!\!+ [8, 7, 9]$$

$$[2, 3, 8, 7, 9] \xrightarrow{(1 ::)} [1, 2, 3, 8, 7, 9]$$

Indução em listas

Θ. $(\forall xs, ys : L\ Nat)[\,\mathrm{sum}\ (xs +\!\!+\ ys) = \mathrm{sum}\ xs + \mathrm{sum}\ ys\,]$

Sejam xs, ys : L Nat.

Por indução no xs.

```
match xs with
   []      ⇝ ....
   (k::ks) ⇝ ....
```

DADOS

xs : L Nat

ys : L Nat

ALVO

$\underset{xs}{\ldots} = \underset{xs}{\ldots}$

CASO [] :

Calculamos:

$\mathrm{sum}\ ([\,] +\!\!+\ ys)$

$= \mathrm{sum}\ ys \quad [\,(+\!\!+).1 \quad ys := ys\,]$

$\mathrm{sum}\ [\,] + \mathrm{sum}\ ys$

$= 0 + \mathrm{sum}\ ys \quad [\,\mathrm{sum}.1\,]$

$= \mathrm{sum}\ ys \quad [\,(id.L) \quad n := \mathrm{sum}\ ys\,]$

$(\forall n : Nat)[\,0 + n = n\,]$

DADOS

[ ] : L Nat

ys : L Nat

ALVO

$\underbrace{\underset{[\,]}{\ldots} = \underset{[\,]}{\ldots}}_{\varphi([\,])}$

a conclusão deste cálculo é: $\boxed{\mathrm{sum}\ ([\,] +\!\!+\ ys) = \mathrm{sum}\ [\,] + \mathrm{sum}\ ys}$

CASO $(k :: ks)$:

Calculamos:

$$
\begin{aligned}
& \mathrm{sum}\ ((k :: ks) \mathbin{+\!\!+} ys) \\
&= \mathrm{sum}\ (k :: (ks \mathbin{+\!\!+} ys)) && [(\mathbin{+\!\!+}).2 \ \ldots] \\
&= k + \mathrm{sum}\ (ks \mathbin{+\!\!+} ys) && [\mathrm{sum}.2 \ \ldots] \\
&= k + (\mathrm{sum}\ ks + \mathrm{sum}\ ys) && [\mathrm{H.I.}] \\
&= (k + \mathrm{sum}\ ks) + \mathrm{sum}\ ys && [(+)\text{-ass}] \\
&= \mathrm{sum}\ (k :: ks) + \mathrm{sum}\ ys && [\mathrm{sum}.2^{\leftarrow}]
\end{aligned}
$$

DADOS

$(k :: ks) : L\ \mathrm{Nat}$

$ys : L\ \mathrm{Nat}$

$$\underbrace{\underset{ks}{\ldots\ldots} = \underset{ks}{\ldots\ldots}}_{\varphi(ks)}$$

H.I.

ALVO

$$\underbrace{\underset{(k::ks)}{\ldots\ldots} = \underset{(k::ks)}{\ldots\ldots}}_{\varphi(k::ks)}$$

Indução em listas

[] preserva φ

$\_::\_$ preserva φ

$(\_::\_) : \alpha \to L\alpha \to L\alpha$

$$\frac{\varphi([\,]) \qquad (\forall ks : L\alpha)\left[\varphi(ks) \Longrightarrow \overbrace{(\forall k : \alpha)\left[\varphi(k :: ks)\right]}^{\text{todas as construções que usam a lista ks são legais}}\right]}{(\forall \ell : L\alpha)\left[\varphi(\ell)\right]}\ \mathrm{IND}_{\varphi}^{L\alpha}$$

Pointwise

lec11

2024-12-02

pointwise

```
pwAdd : L Nat → L Nat → L Nat
pwAdd [] _ = []
pwAdd _ [] = []
pwAdd (n::ns) (m::ms) = (n + m) :: pwAdd ns ms
```

pwAdd

[1, 2, 0, 7]

[3, 5, 2]

= [4, 7, 2]

trocando a ordem das equações dá para economizar:

```
pwAdd (n::ns) (m::ms) = (n + m) :: pwAdd ns ms
pwAdd _ _ = []
```

pwAdd [1,2,3] [4,5]

= (1+4) :: pwAdd [2,3] [5]

⋮

```
pwMult (n::ns) (m::ms) = (n · m) :: pwMult ns ms
pwMult _ _ = []
```

Abstraindo ...

```
pwAdd  (n::ns) (m::ms) = (n + m) :: pwAdd ns ms
pwAdd  _       _       = []

pwMult (n::ns) (m::ms) = (n · m) :: pwMult ns ms
pwMult _       _       = []


pw : (Nat → Nat → Nat) → L Nat → L Nat → L Nat
pw ♡  (n :: ns) (m :: ms) = (n ♡ m) :: pw ♡ ns ms
pw ♡  _         _         = []
pw op (n :: ns) (m :: ms) = op n m :: pw op ns ms
```

$$\mathrm{pwAdd} \stackrel{\mathrm{def}}{=} \mathrm{pw}\ (+)$$

$$\mathrm{pwMult} \stackrel{\mathrm{def}}{=} \mathrm{pw}\ (\cdot)$$

$$\begin{aligned}
&\mathrm{pwAdd}\ [1,2,3]\ [4,5] \\
&= (\mathrm{pw}\ (+))\ [1,2,3]\ [4,5] \\
&= \mathrm{pw}\ (+)\ [1,2,3]\ [4,5] \\
&= (1+4) :: \mathrm{pw}\ (+)\ [2,3]\ [5] \\
&\ \vdots
\end{aligned}$$

... mais e mais ...

$$(\alpha \to \beta \to \gamma) \to L\ \alpha \to L\ \beta \to L\ \gamma$$

$$\mathrm{pw} : (\alpha \to \alpha \to \alpha) \to L\ \alpha \to L\ \alpha \to L\ \alpha$$

$$\mathrm{pw}\ \underset{f}{op}\ (x :: xs)\ (y :: ys) = \underset{f}{op}\ x\ y :: \mathrm{pw}\ \underset{f}{op}\ xs\ ys$$

$$\mathrm{pw}\ \underset{f}{op}\ \_\ \_ = [\,]$$

NUNCA!

!! $(b == True) == True$

$b == True \rightsquigarrow b$

$b == False \rightsquigarrow not\ b$

$if\ b\ then\ True\ else\ False \rightsquigarrow b$

$if\ b\ then\ False\ else\ True \rightsquigarrow not\ b$

$if\ b\ then\ True\ else\ b' \rightsquigarrow b\ ou\ b'$

⋮

HW!

all & any

$$\begin{aligned}
&\text{allEven} : L\ \text{Nat} \to \text{Bool}\\
&\text{allEven}\ [\,] = \text{True}\\
&\text{allEven}\ (n :: ns) = \text{ev}\ n \ \overset{\&}{\underset{\&\&}{\wedge}}\ \text{allEven}\ ns
\end{aligned}$$

$$\begin{aligned}
&\text{all} : (\alpha \to \text{Bool}) \to L\,\alpha \to \text{Bool}\\
&\text{all}\ p\ [\,] = \text{True}\\
&\text{all}\ p\ (x :: xs) = p\ x \wedge \text{all}\ p\ xs
\end{aligned}$$

$$\begin{aligned}
&\text{all ev}\ [2,4,6]\\
&= \text{ev}\ 2 \wedge \text{all ev}\ [4,6]\\
&= \text{ev}\ 2 \wedge \text{ev}\ 4 \wedge \text{all ev}\ [6]\\
&= \text{ev}\ 2 \wedge \text{ev}\ 4 \wedge \text{ev}\ 6 \wedge \text{all ev}\ [\,]\\
&= \text{True} \wedge \text{all ev}\ [\,]\\
&= \text{True} \wedge \text{True}
\end{aligned}$$

$$\text{allEven} = \text{all even}$$

$$\begin{aligned}
&\text{any} : (\alpha \to \text{Bool}) \to L\,\alpha \to \text{Bool}\\
&\text{any}\ p\ [\,] = \text{False}\\
&\text{any}\ p\ (x :: xs) = p\ x \vee \text{any}\ p\ xs
\end{aligned}$$

fold

$and : L\ Bool \to Bool$

$and\ [\,] = True$

$and\ (b :: bs) = b \wedge and\ bs$

$sum : L\ Nat \to Nat$

$sum\ [\,] = 0$

$sum\ (n :: ns) = n + sum\ ns$

$fold : (\alpha \to \alpha \to \alpha) \to \alpha \to L\,\alpha \to \alpha$

i, z, ...

$fold\ op\ e\ [\,] = e$

$fold\ op\ e\ (x :: xs) = op\ x\ (fold\ op\ e\ xs)$

ou, usando let-in:

$fold\ op\ e\ (x :: xs) = \text{let}\ v = fold\ op\ e\ xs$

$\text{in}\ op\ x\ v$

let-in expressions

```
let  x = 2
     y = x + 1
 in  x · y + 3
```

$: \mathrm{Nat}$

euclid : $\mathrm{Nat} \times \mathrm{Nat} \to \mathrm{Nat} \times \mathrm{Nat}$

```
let
     t = euclid (a,b)
 in
     ... t.l ... t.r ...
```

fst t   snd t

outl t   outr t

pattern matching

```
let
     (q,r) = euclid (a,b)
 in
     ... q ... r ...
```

Unit & Empty

lec12

2024-12-16

```
void print (str s)
int five (void)
```

$\text{print} : \text{str} \to \text{void}$?

$\text{five} : \text{void} \to \text{int}$ $\mathbb{Z}$

aplicação de função (para o programador C)

five ( )

nulária
0-ária
aridade 0

argumentos

$g(a_1, \ldots, a_n)$

n-ária
aridade n

aplicação de função

argumento (sempre um)

five ␣ ( )

produtório de 0 tipos: ?

aridade

g ␣ $(a_1, \ldots, a_n)$

produtório de n tipos: $\alpha_1 \times \cdots \times \alpha_n$

aridade

$$\frac{f : \alpha \to \beta \qquad a : \alpha}{f\,a : \beta}\ (\text{uso}\ \to)$$

$$\frac{a : \alpha \qquad b : \beta}{(a, b) : \alpha \times \beta}\ (\text{construção}\ \_\times\_)$$

$$\frac{a : \alpha \qquad b : \beta \qquad c : \gamma}{(a, b, c) : \alpha \times \beta \times \gamma}\ (\text{construção}\ \_\times\_\times\_)$$

$$\frac{}{() : \mathbb{1}}\ (\text{construção}\ \mathbb{1})$$

$\mathbb{1}$ — Unit : Ty

```
data Unit
    * : Unit
```

()

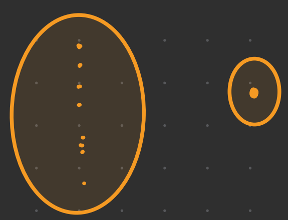
$f : \mathbb{1} \to \beta$

tantas quantos os β's

$g : \alpha \to \mathbb{1}$

exatamente uma

$\mathbb{N} \xrightarrow{!} \mathbb{1}$



$\mathbb{0}$ — Empty : Ty

```
data Empty
```

$h : \mathbb{0} \to \beta$

exatamente uma

caso especial: $\beta := \mathbb{0}$

$k : \alpha \to \mathbb{0}$

nenhuma, caso α tem habitantes

exatamente uma, caso contrário

Aritmética de tipos (teaser)

o que seria isso?

$$|\alpha + \beta| = |\alpha| + |\beta|$$

$$|\alpha \times \beta| = |\alpha| \cdot |\beta|$$

$$|\alpha \to \beta| = \underbrace{|\beta| \cdot \cdots \cdot |\beta|}_{|\alpha| \text{ vezes}} = |\beta|^{|\alpha|}$$

$$|\mathbb{0}| = 0$$

$$|\mathbb{1}| = 1$$

Maybe α

$$\text{head} : L\ \alpha \to \alpha$$

$$\text{head}\ [\,] =$$

$$\text{head}\ (x :: \_) = x$$

$$
\boxed{
\begin{array}{l}
\text{data Maybe } \alpha \\
\quad \text{Nothing} : \text{Maybe } \alpha \\
\quad \text{Just} : \alpha \to \text{Maybe } \alpha
\end{array}
}
$$

$$\text{safeHead} : L\ \alpha \to M\ \alpha$$

$$\text{safeHead}\ [\,] = \text{Nothing}$$

$$\text{safeHead}\ (x :: \_) = \text{Just}\ x$$

$$\text{find} : \alpha \to L\ \alpha \to \overset{(\text{Int})}{\text{Nat}}$$

$$\text{index} : \text{Nat} \to L\ \alpha \to \alpha$$

$$\text{Maybe} : \text{Ty} \to \text{Ty}$$

$$\text{Maybe Nat} : \text{Ty}$$

$$\text{Maybe (List Int)} : \text{Ty}$$

$$\vdots$$

$$\text{index} : \text{Nat} \to L\ \alpha \to M\ \alpha$$

$$\text{index}\ \_\ [\,] = \text{Nothing}$$

$$\text{index}\ 0\ (x :: xs) = \text{Just}\ x$$

$$\text{index}\ (S\ n)\ (x :: xs) = ?$$

Sum types: Either α β

Python: [42, 3, True, 2, False, False]

~~data NatOrBool~~

~~N : Nat → NOB~~

~~B : Bool → NOB~~

(E Nat Bool)

(Nat + Bool)

melhor: [N 42, N 3, B True, N 2, B False, ..] : L ~~NOB~~

$\alpha + \beta$

data Either α β

L : $\alpha \to$ E α β

R : $\beta \to$ E α β

Either : Ty → Ty → Ty

Either Nat Bool : Ty

Either (M Bool) String : Ty

Either (L Nat) : Ty → Ty

⋮