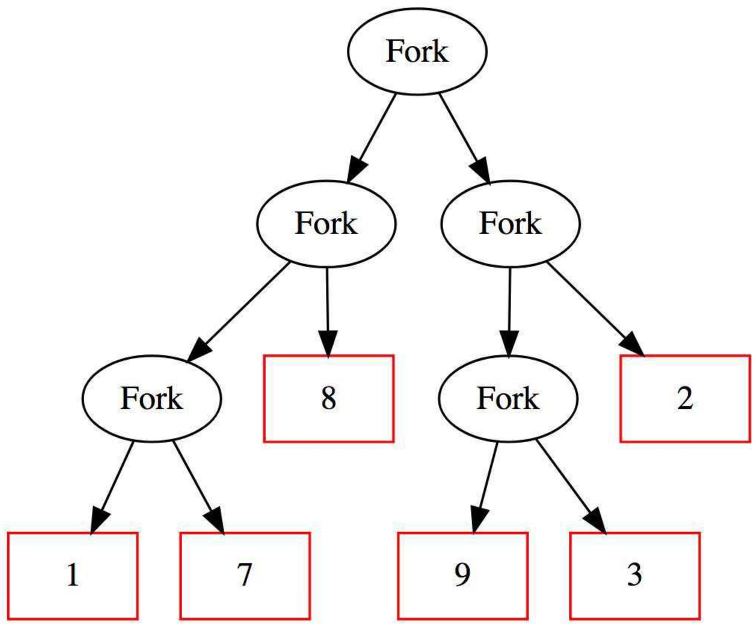
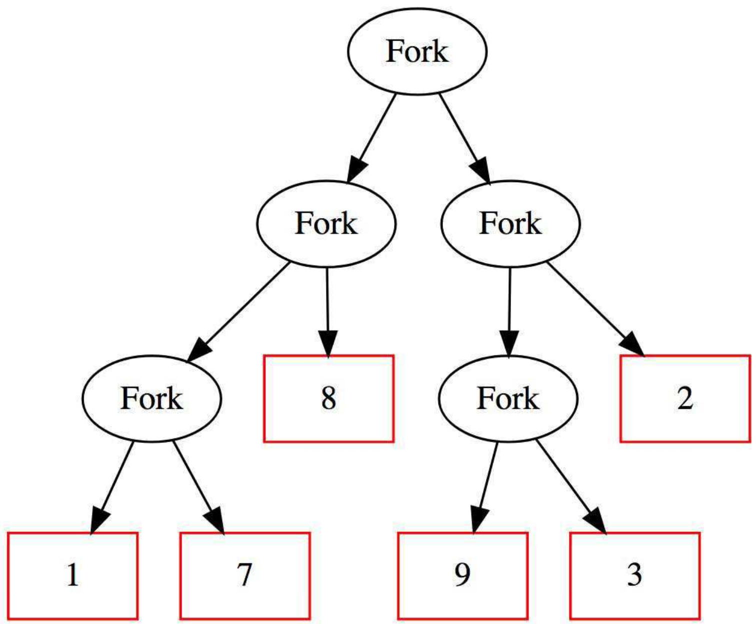
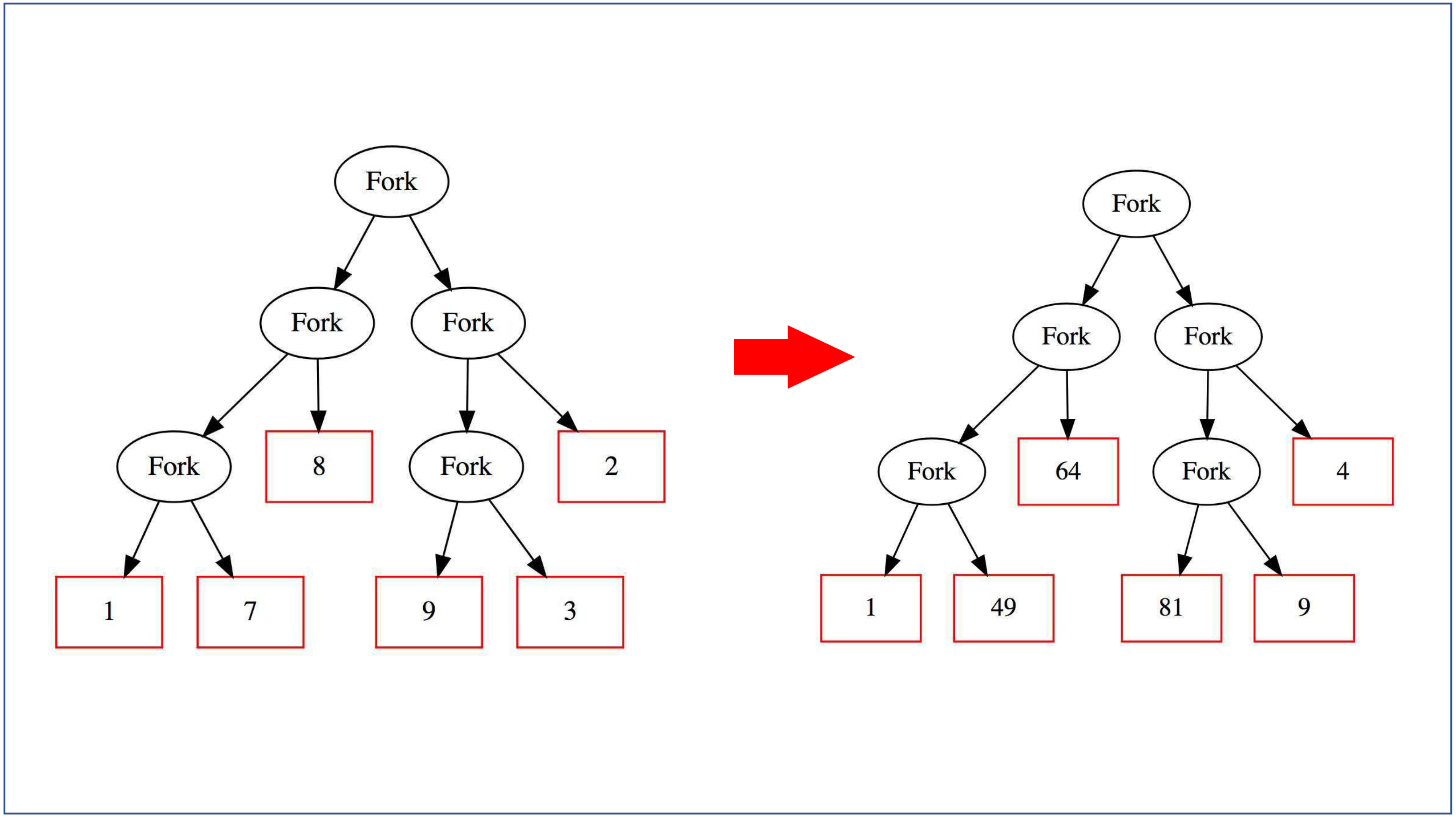
Cálculo de Programas
Aula T09

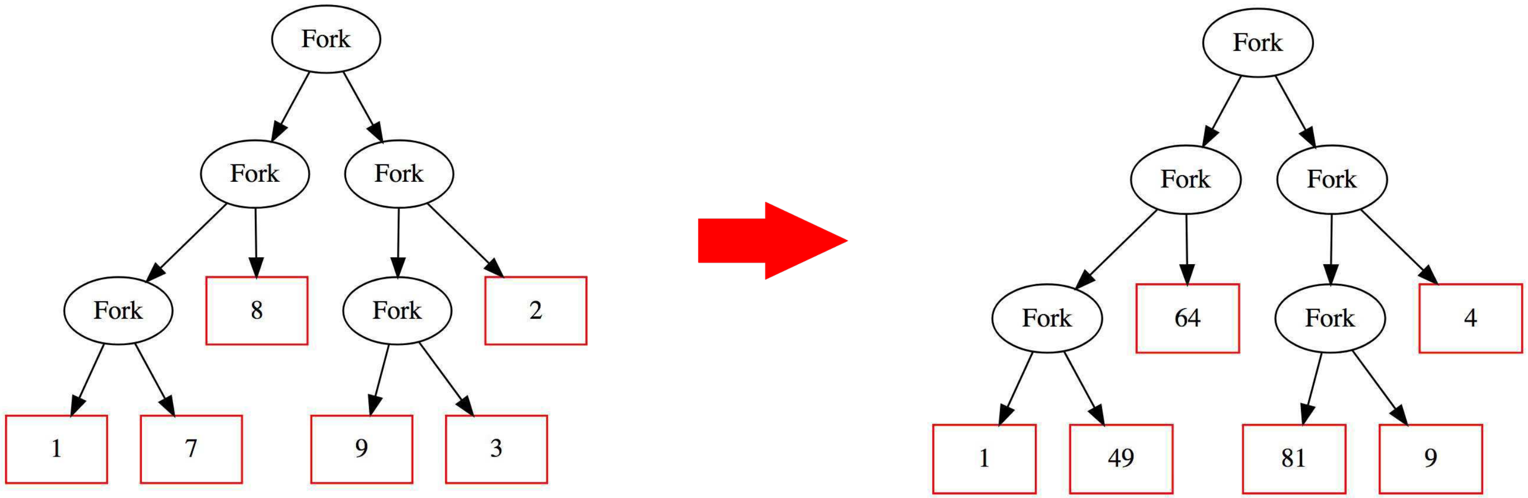



```
data LTree = Leaf Int | Fork (LTree, LTree)
```



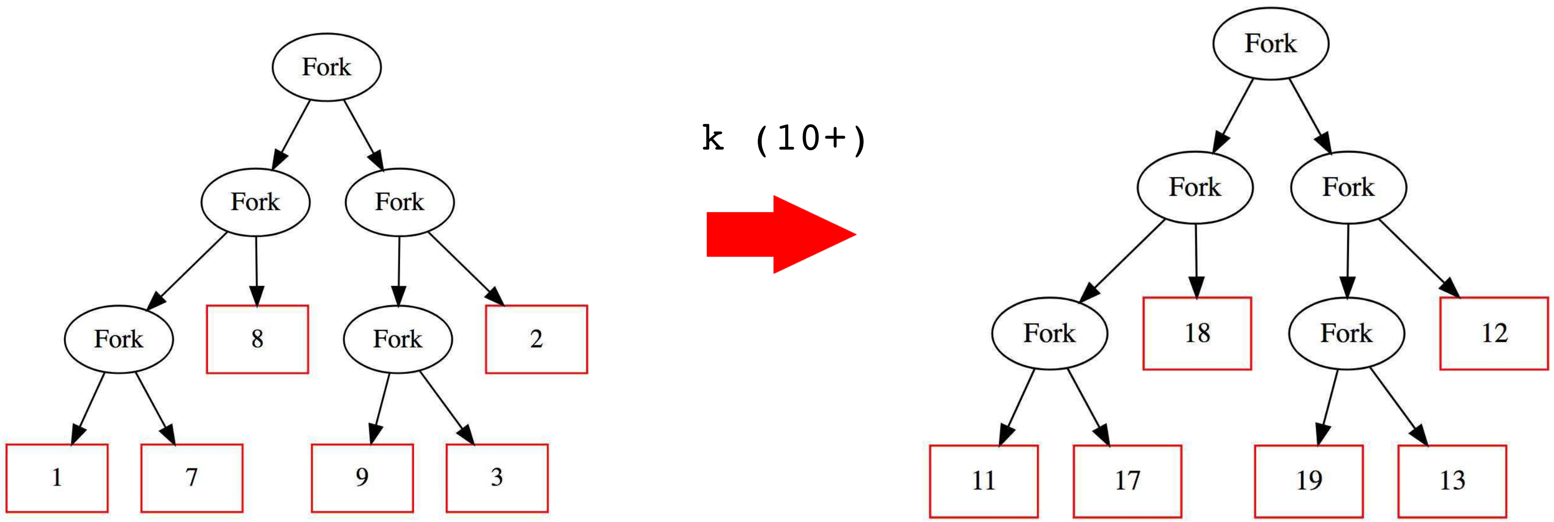

```
data LTree = Leaf Int | Fork (LTree, LTree)
```


Fork
Fork
Fork
Fork
8
Fork
2
1
7
9
3
Fork
Fork
Fork
Fork
64
Fork
4
1
49
81
9




```haskell
data LTree = Leaf Int | Fork (LTree, LTree)

k (Leaf x) = Leaf (x^2)
k (Fork (l,r)) = Fork (k l, k r)
```

```haskell
k :: (Int -> Int) -> LTree -> LTree
k f (Leaf x) = Leaf (f x)
k f (Fork (l,r)) = Fork (k f l, k f r)
```



```haskell
k :: (Int -> Int) -> LTree -> LTree
k f (Leaf x) = Leaf (f x)
k f (Fork (l,r)) = Fork (k f l, k f r)
```

$$\begin{cases} k\ f \cdot Leaf = Leaf \cdot f \\ k\ f \cdot Fork = Fork \cdot (k\ f \times k\ f) \end{cases}$$

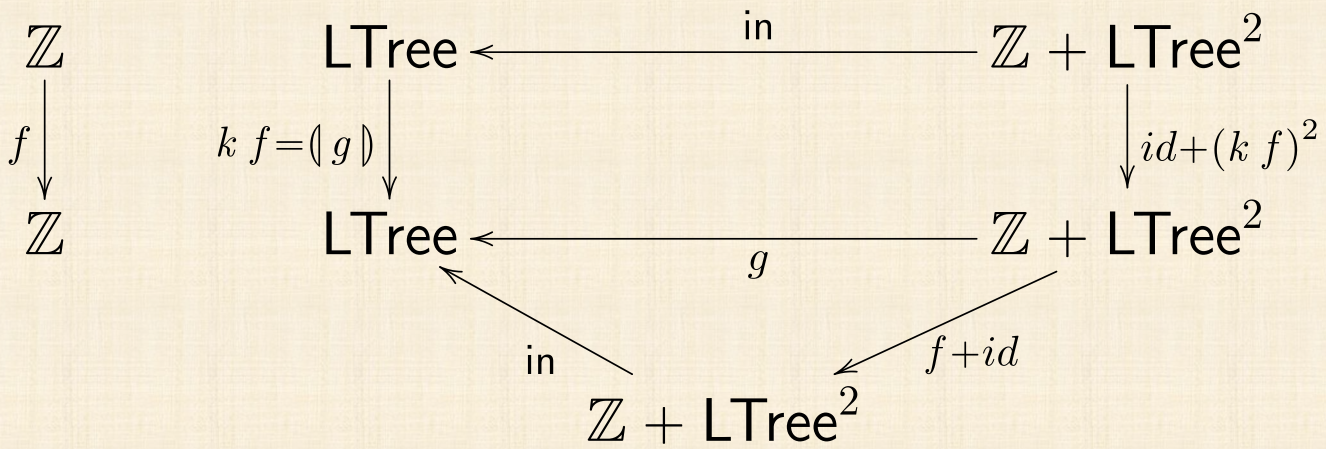
$$\equiv \qquad \{ \text{ Eq-+ ; fusão-+ ; absorção-+ } \}$$

$$k\ f \cdot [Leaf\ , Fork] = [Leaf\ , Fork] \cdot (f + k\ f \times k\ f)$$

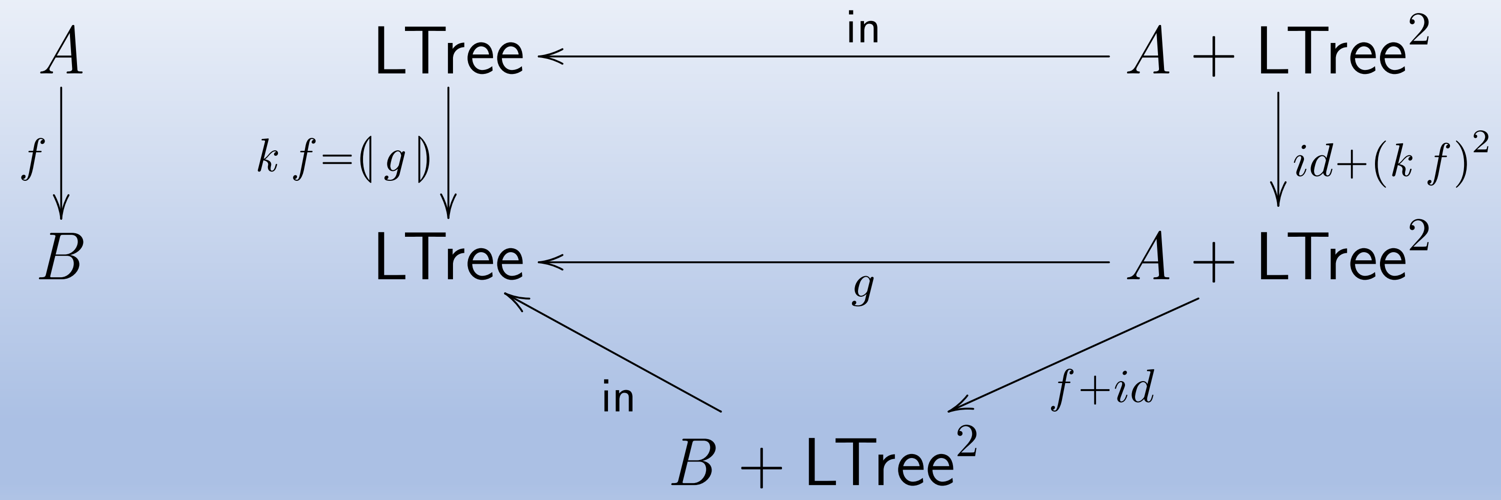
$$\equiv \qquad \{ \ [Leaf\ , Fork] = \mathsf{in} \text{ ; functor-+ } \}$$

$$k\ f \cdot \mathsf{in} = \mathsf{in} \cdot (f + id) \cdot (id + k\ f \times k\ f)$$

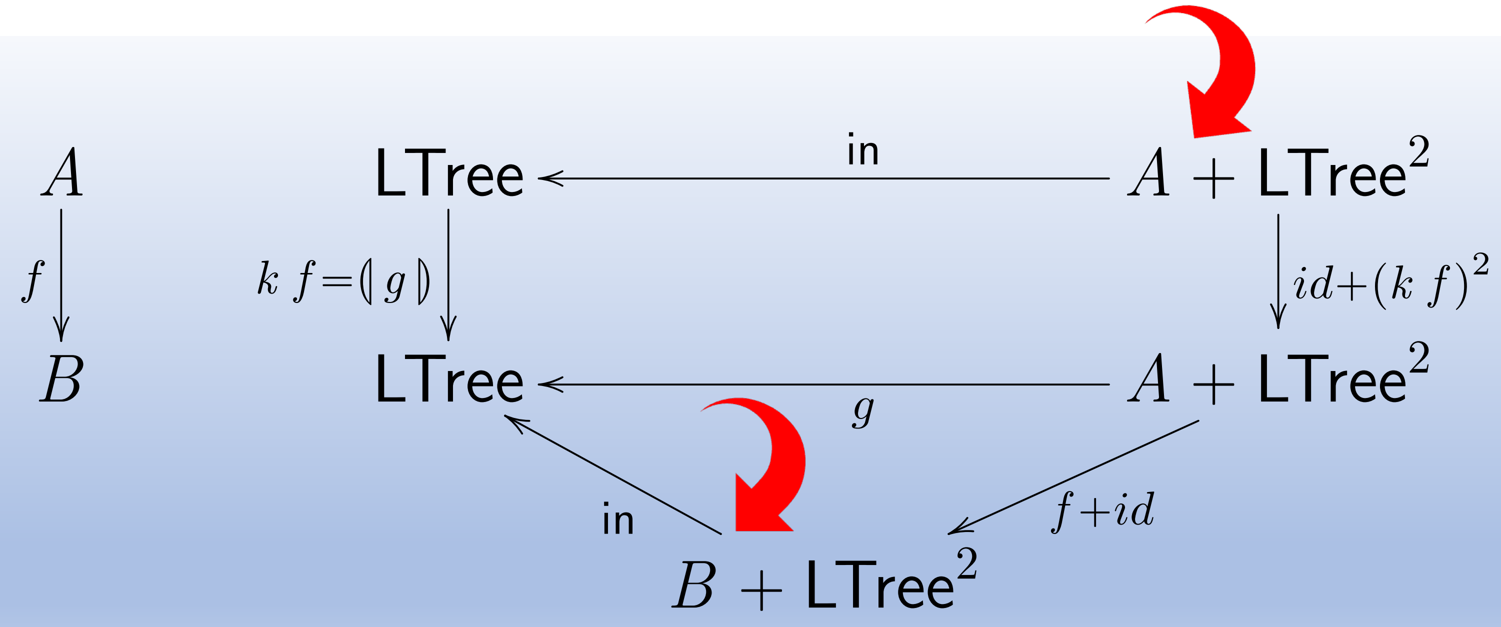
$$k\ f \cdot \mathsf{in} = \mathsf{in} \cdot (f + id) \cdot (id + k\ f \times k\ f)$$



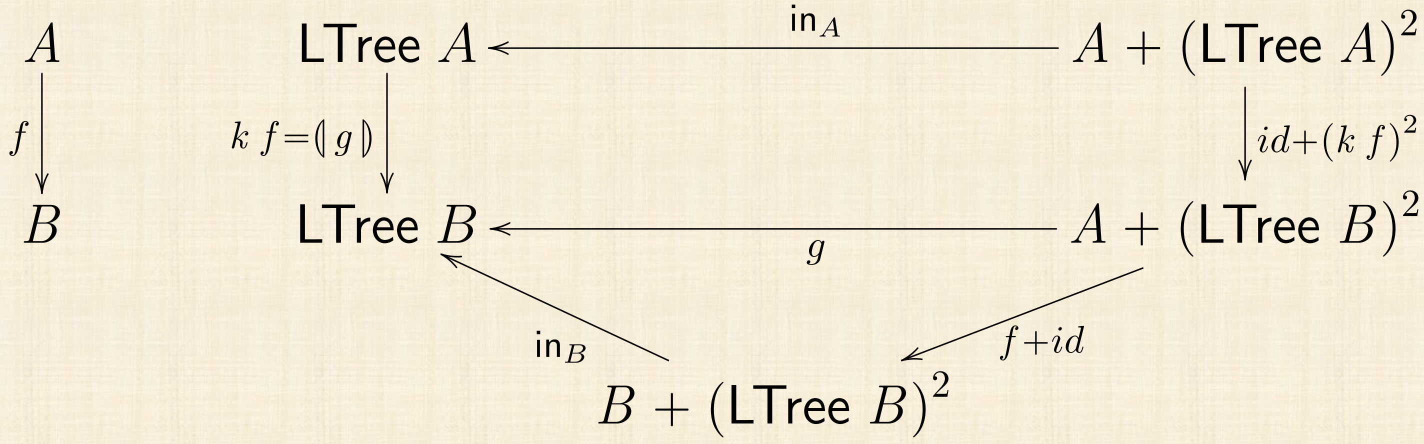
$$k\ f \cdot \mathsf{in} = \mathsf{in} \cdot (f + id) \cdot (id + k\ f \times k\ f)$$



$$k\ f \cdot \mathsf{in} = \mathsf{in} \cdot (f + id) \cdot (id + k\ f \times k\ f)$$



$$k\ f = (\!|\, \mathsf{in} \cdot (f + id) \,|\!)$$



$$k\ f = (\!|\, \mathsf{in} \cdot (f + id) \,|\!)$$

$$k\ f = (\!|\, \mathsf{in} \cdot (f + id) \,|\!)$$

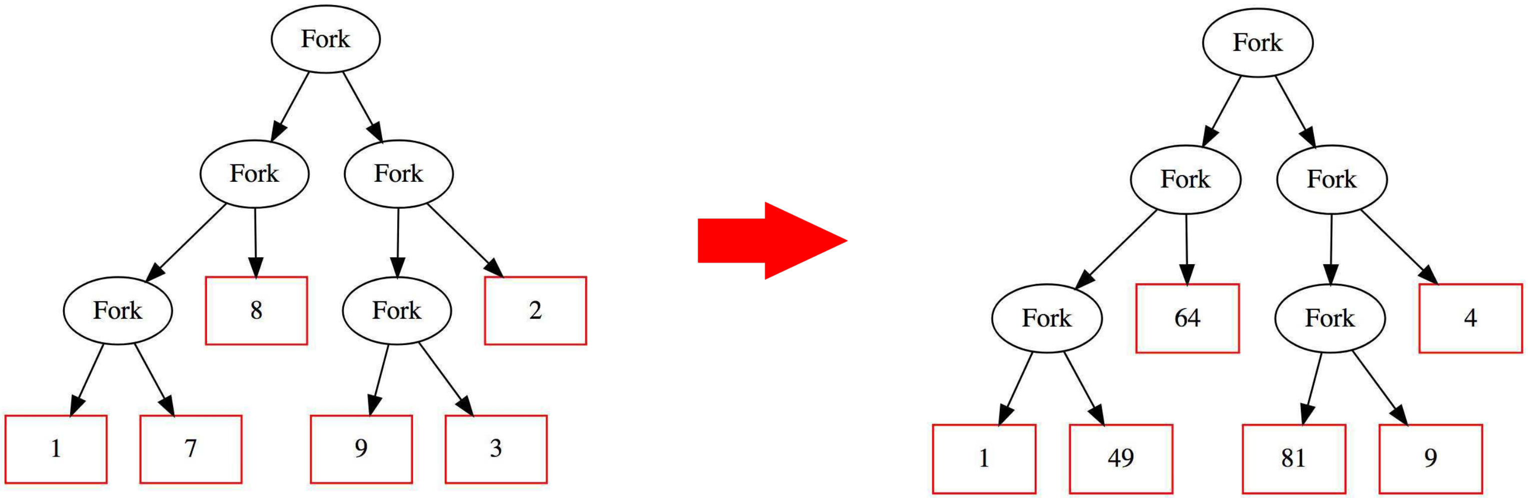
$$k\ id = (\!|\, \mathsf{in} \,|\!) = id$$

$$
\begin{array}{ll}
& (\!|\, g \,|\!) \cdot (\!|\, \mathsf{in} \cdot (f + id) \,|\!) = (\!|\, g \cdot (f + id) \,|\!) \\
\Leftarrow & \qquad \{ \text{ fusão-cata } \} \\
& (\!|\, g \,|\!) \cdot \mathsf{in} \cdot (f + id) = g \cdot (f + id) \cdot (id + (\!|\, g \,|\!)^2) \\
\equiv & \qquad \{ \text{ cancelamento-cata } \} \\
& g \cdot (id + (\!|\, g \,|\!)^2) \cdot (f + id) = g \cdot (f + id) \cdot (id + (\!|\, g \,|\!)^2) \\
\equiv & \qquad \{ \text{ functor-}+\text{ duas vezes; natural-}id\text{ quatro vezes } \} \\
& g \cdot (f + (\!|\, g \,|\!)^2) = g \cdot (f + (\!|\, g \,|\!)^2) \\
\equiv & \qquad \{ \text{ trivial } \} \\
& true
\end{array}
$$

$$
\begin{array}{ll}
& k\ (f \cdot g) \\
= & \{\ k\ f = (\!|\, \mathsf{in} \cdot (f + id) \,|\!)\ \} \\
& (\!|\, \mathsf{in} \cdot (f \cdot g + id) \,|\!) \\
= & \{\ \text{functor-+ etc}\ \} \\
& (\!|\, \mathsf{in} \cdot (f + id) \cdot (g + id) \,|\!) \\
= & \{\ \text{absorção-cata}\ \} \\
& (\!|\, \mathsf{in} \cdot (f + id) \,|\!) \cdot (\!|\, \mathsf{in} \cdot (g + id) \,|\!) \\
= & \{\ k\ f = (\!|\, \mathsf{in} \cdot (f + id) \,|\!)\ \text{duas vezes}\ \} \\
& k\ f \cdot k\ g
\end{array}
$$

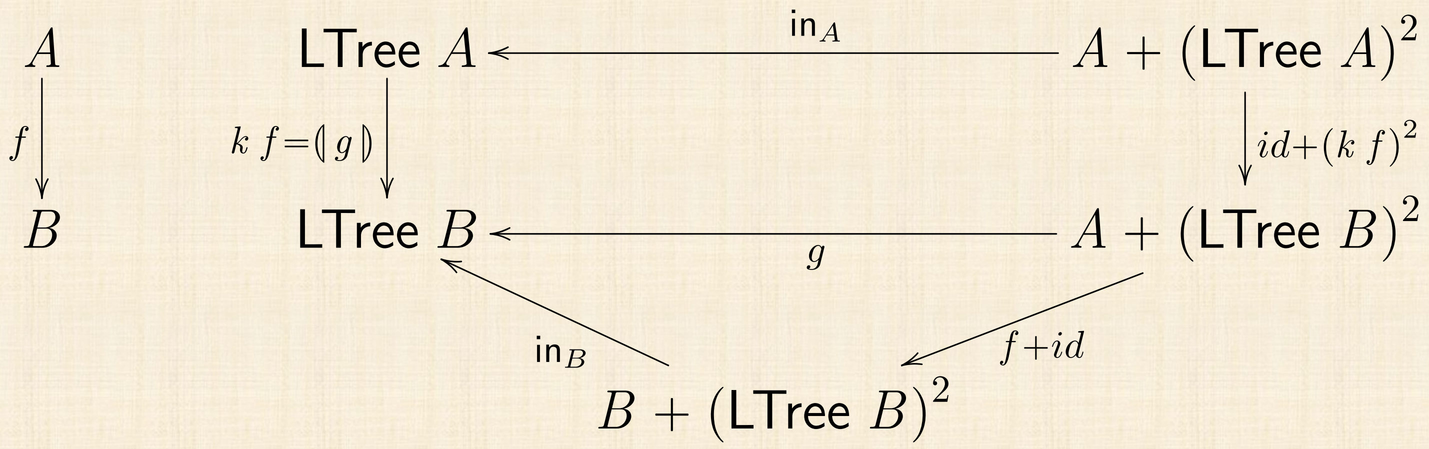
$$k\ id = (\!|\, \mathsf{in} \,|\!) = id$$

$$k\ (f \cdot g) = k\ f \cdot k\ g$$

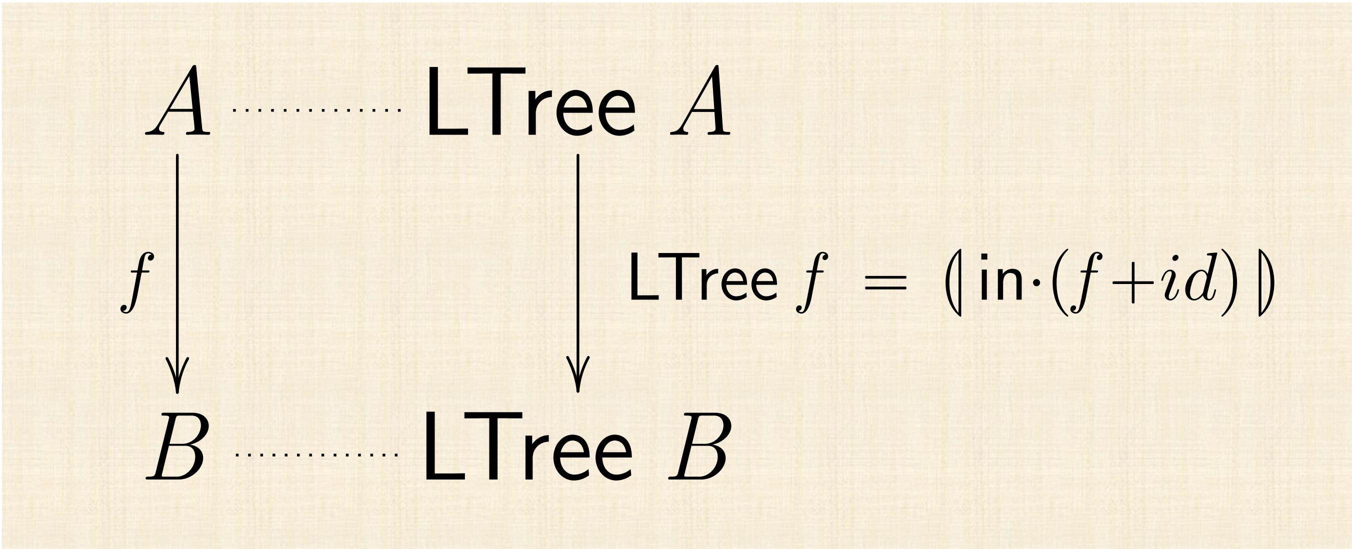



```haskell
data LTree = Leaf Int | Fork (LTree, LTree)

k (Leaf x) = Leaf (x^2)
k (Fork (l,r)) = Fork (k l, k r)
```

$$k\ f = (\!|\, \mathsf{in} \cdot (f + id) \,|\!)$$



FUNCTOR DO TIPO LTREE



FUNCTOR DO TIPO LTREE

$$\mathsf{LTree}\ A \underset{\mathsf{in}}{\overset{\mathsf{out}}{\cong}} \underbrace{A + (\mathsf{LTree}\ A)^2}_{\mathbf{F}\ (\mathsf{LTree}\ A)}$$

$$\mathsf{LTree}\ B \underset{\mathsf{in}}{\overset{\mathsf{out}}{\cong}} \underbrace{B + (\mathsf{LTree}\ B)^2}_{\mathbf{F}\ (\mathsf{LTree}\ B)}$$

FUNCTOR DO TIPO LTREE

$$\mathsf{F}\ X = A + X^2 \qquad ?$$

$$\mathsf{F}\ X = B + X^2 \qquad ?$$

$$\mathsf{LTree}\ A \underset{\mathsf{in}}{\overset{\mathsf{out}}{\cong}} \underbrace{A + (\mathsf{LTree}\ A)^2}_{\mathsf{F}\ (\mathsf{LTree}\ A)}$$

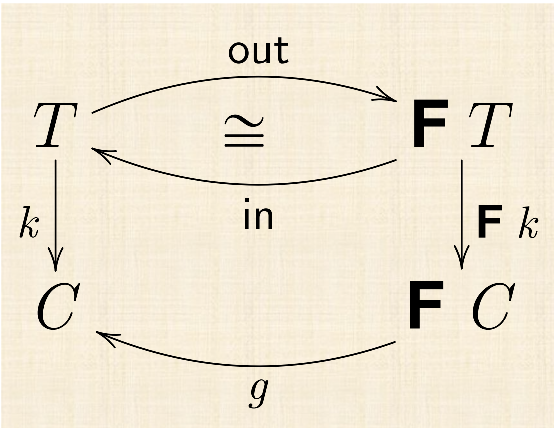
$$\mathsf{LTree}\ B \underset{\mathsf{in}}{\overset{\mathsf{out}}{\cong}} \underbrace{B + (\mathsf{LTree}\ B)^2}_{\mathsf{F}\ (\mathsf{LTree}\ B)}$$

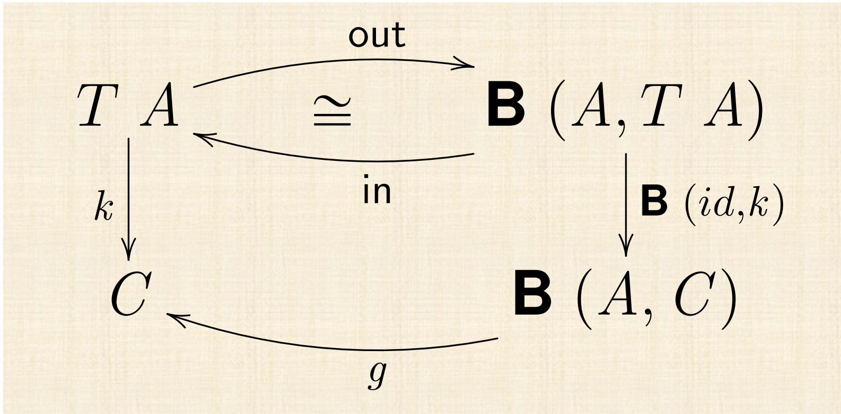
BIFUNCTOR DO TIPO LTREE

$$\mathsf{LTree}\ X \underset{\mathsf{in}}{\overset{\mathsf{out}}{\rightleftarrows}} \cong \underbrace{X + (\mathsf{LTree}\ X)^2}_{\mathbf{B}\ (X, \mathsf{LTree}\ X)}$$

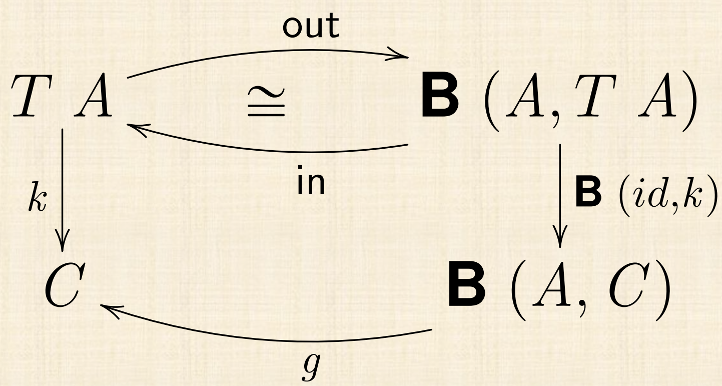
$$\mathbf{B}\ (X, Y) = X + Y^2$$

CATAMORFISMOS (GENERALIZAÇÃO)





CATAMORFISMOS (CASO GERAL)



Functor do tipo:

$$T\ f = (\!|\ \mathsf{in} \cdot \mathbf{B}\ (f, id)\ |\!)$$

Abreviatura:

$$\mathbf{F}\ k = \mathbf{B}\ (id, k)$$

Propriedade universal

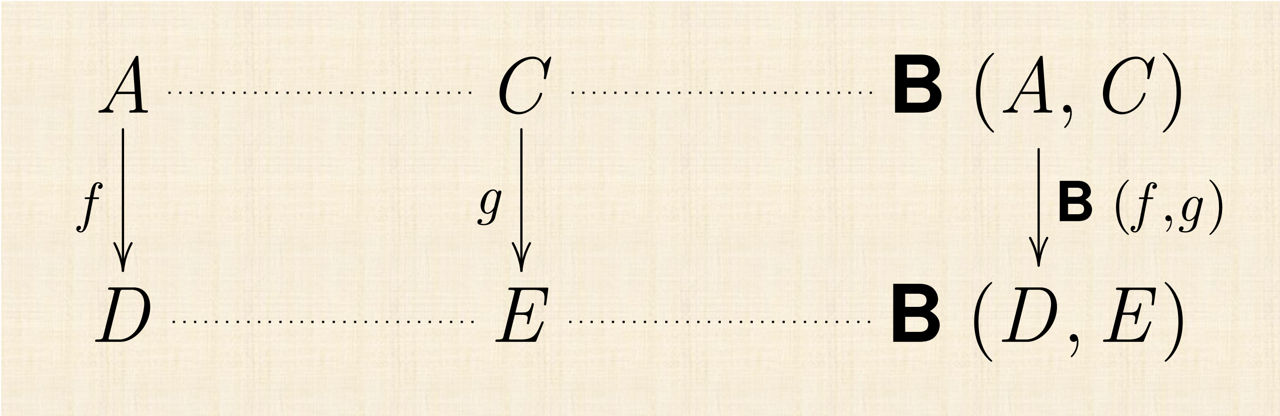
$$k = (\!|\ g\ |\!) \Leftrightarrow k \cdot \mathsf{in} = g \cdot \mathbf{B}\ (id, k)$$

(Apontamentos: páginas 97-101)

CATAMORFISMOS (LEIS)

Universal-cata	$k = (\| g \|) \quad \Leftrightarrow \quad k \cdot \mathsf{in} = g \cdot \mathsf{F}\, k$	(43)
Cancelamento-cata	$(\| g \|) \cdot \mathsf{in} = g \cdot \mathsf{F}\, (\| g \|)$	(44)
Reflexão-cata	$(\| \mathsf{in} \|) = id_{\mathsf{T}}$	(45)
Fusão-cata	$f \cdot (\| g \|) = (\| h \|) \quad \Leftarrow \quad f \cdot g = h \cdot \mathsf{F}\, f$	(46)
Base-cata	$\mathsf{F}\, f \quad = \quad \mathsf{B}\ (id, f)$	(47)
Def-map-cata	$\mathsf{T}\, f \quad = \quad (\| \mathsf{in} \cdot \mathsf{B}(f, id) \|)$	(48)
Absorção-cata	$(\| g \|) \cdot \mathsf{T}\, f \quad = \quad (\| g \cdot \mathsf{B}(f, id) \|)$	(49)

BIFUNCTORES



BIFUNCTORES (LEIS)

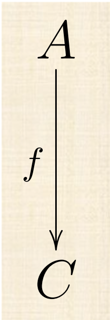
$$
\begin{array}{ccccc}
A & \cdots\cdots & C & \cdots\cdots & \mathbf{B}\,(A, C) \\
{\scriptstyle f}\downarrow & & {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle \mathbf{B}\,(f,g)} \\
D & \cdots\cdots & E & \cdots\cdots & \mathbf{B}\,(D, E)
\end{array}
$$

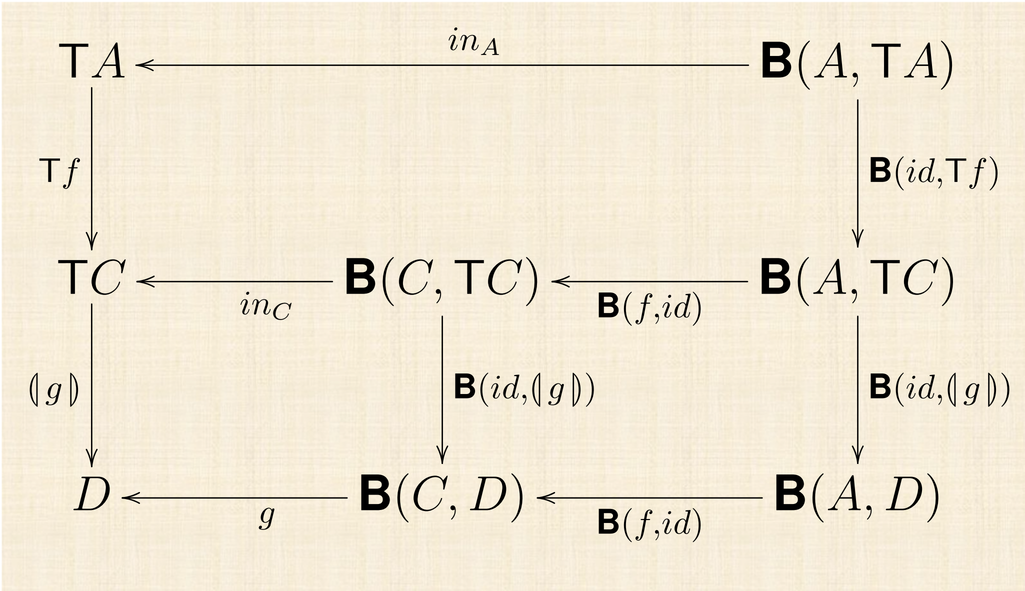
$$\mathbf{B}\,(id, id) = id$$

$$\mathbf{B}\,(h \cdot f, k \cdot g) = \mathbf{B}\,(h, k) \cdot \mathbf{B}\,(f, g)$$

ABSORÇÃO-CATA

$$(\!| g |\!) \cdot \mathsf{T} f = (\!| g \cdot \mathbf{B}\,(f, id) |\!)$$





(a) Árvores com informação de tipo A nos nós:

$$\mathsf{T} = \mathsf{BTree}\ A \qquad \begin{cases} \mathsf{F}\ X = 1 + A \times X^2 \\ \mathsf{F}\ f = id + id \times f^2 \end{cases} \qquad \mathsf{in} = [\underline{Empty}\ , Node]$$

Haskell: **data** $\mathsf{BTree}\ a = Empty \mid Node\ (a, (\mathsf{BTree}\ a, \mathsf{BTree}\ a))$

(b) Árvores com informação de tipo A nas folhas:

$$\mathsf{T} = \mathsf{LTree}\ A \qquad \begin{cases} \mathsf{F}\ X = A + X^2 \\ \mathsf{F}\ f = id + f^2 \end{cases} \qquad \mathsf{in} = [Leaf\ , Fork]$$

Haskell: **data** $\mathsf{LTree}\ a = Leaf\ \ a \mid Fork\ (\mathsf{LTree}\ a, \mathsf{LTree}\ a)$

(c) Árvores com informação nos nós e nas folhas:

$$\mathsf{T} = \mathsf{FTree}\ B\ \ A \qquad \begin{cases} \mathsf{F}\ X = B + A \times X^2 \\ \mathsf{F}\ f = id + id \times f^2 \end{cases} \qquad \mathsf{in} = [Unit\ , Comp]$$

Haskell: **data** $\mathsf{FTree}\ b\ \ a = Unit\ \ b \mid Comp\ (a, (\mathsf{FTree}\ b\ \ a, \mathsf{FTree}\ b\ \ a))$

(d) Árvores de expressão:

$$\mathsf{T} = Expr\ \ V\ \ O \qquad \begin{cases} \mathsf{F}\ X = V + O \times X^* \\ \mathsf{F}\ f = id + id \times \mathsf{map}\ f \end{cases} \qquad \mathsf{in} = [Var\ , Op]$$

Haskell: **data** $Expr\ \ v\ \ o = Var\ \ v \mid Op\ (o, [Expr\ \ v\ \ o])$

(a) Árvores com informação de tipo A nos nós:

$$\mathsf{T} = \mathsf{BTree}\ X \qquad \begin{cases} \mathsf{B}\,(X, Y) = 1 + X \times Y^2 \\ \mathsf{B}\,(f, g) = id + f \times g^2 \end{cases} \qquad \mathsf{in} = [\underline{Empty}\ , Node]$$

Haskell: **data** $\mathsf{BTree}\ a = Empty \mid Node\ (a, (\mathsf{BTree}\ a, \mathsf{BTree}\ a))$

(b) Árvores com informação de tipo A nas folhas:

$$\mathsf{T} = \mathsf{LTree}\ X \qquad \begin{cases} \mathsf{B}\,(X, Y) = X + Y^2 \\ \mathsf{B}\,(f, g) = f + g^2 \end{cases} \qquad \mathsf{in} = [Leaf\ , Fork]$$

Haskell: **data** $\mathsf{LTree}\ a = Leaf\ a \mid Fork\ (\mathsf{LTree}\ a, \mathsf{LTree}\ a)$

(c) Árvores com informação nos nós e nas folhas:

$$\mathsf{T} = \mathsf{FTree}\ Z\ X \qquad \begin{cases} \mathsf{B}\,(Z, X, Y) = Z + X \times Y^2 \\ \mathsf{B}\,(h, f, g) = h + f \times g^2 \end{cases} \qquad \mathsf{in} = [Unit\ , Comp]$$

Haskell: **data** $\mathsf{FTree}\ b\ a = Unit\ b \mid Comp\ (a, (\mathsf{FTree}\ b\ a, \mathsf{FTree}\ b\ a))$

(d) Árvores de expressão:

$$\mathsf{T} = Expr\ Z\ X \qquad \begin{cases} \mathsf{B}\,(Z, X, Y) = Z + X \times Y^* \\ \mathsf{B}\,(h, f, g) = h + f \times \mathsf{map}\ g \end{cases} \qquad \mathsf{in} = [Var\ , Op]$$

Haskell: **data** $Expr\ v\ o = Var\ v \mid Op\ (o, [Expr\ v\ o])$

```haskell
data Rose a = Rose a [Rose a] deriving Show

inRose = uncurry Rose

outRose (Rose a x) = (a,x)
```

$$\mathsf{Rose}\ A \;\underset{\mathsf{in}}{\overset{\mathsf{out}}{\rightleftarrows}}\; A \times (\mathsf{Rose}\ A)^* \qquad \mathsf{Rose}\ A \cong A \times (\mathsf{Rose}\ A)^*$$

$$\mathsf{Rose}\ A \underset{\mathsf{in}}{\overset{\mathsf{out}}{\cong}} A \times (\mathsf{Rose}\ A)^*$$

$$\mathbf{B}\,(X, Y) = X \times Y^*$$

$$\mathbf{B}\,(f, g) = f \times g^*$$

$$A \times Y^*$$

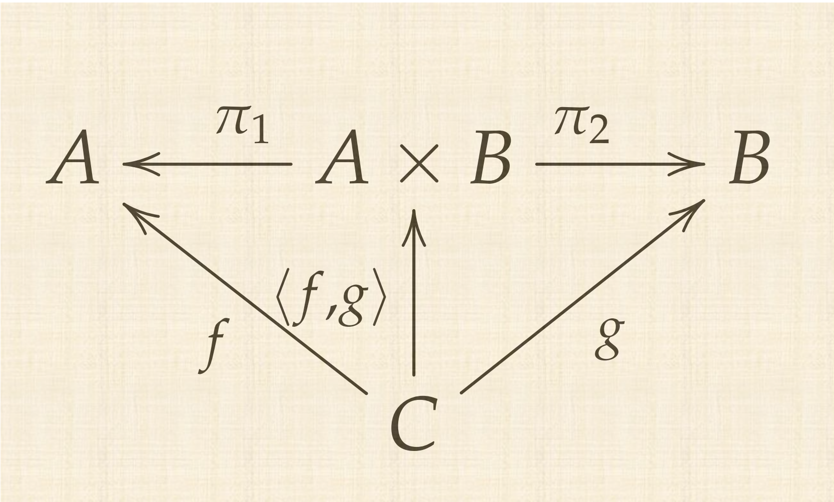
$$X \times Y^*$$

```haskell
data Rose a = Rose a [Rose a] deriving Show

inRose = uncurry Rose

outRose (Rose a x) = (a,x)
```

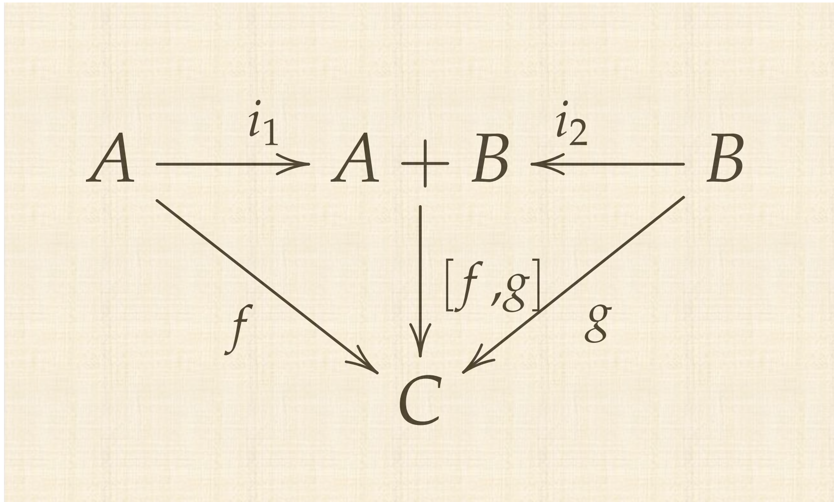
$$\mathsf{B}\,(X, Y) = X \times Y^*$$

$$\mathsf{B}\,(f, g) = f \times g^*$$

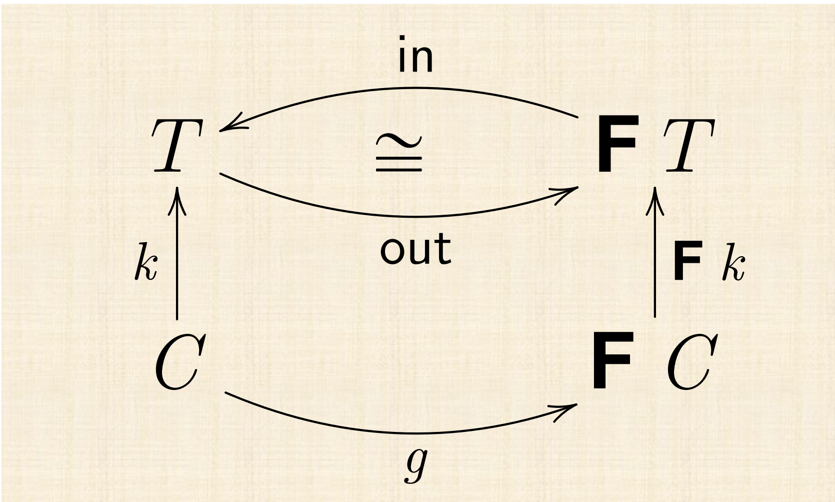
```haskell
f >< map g
```



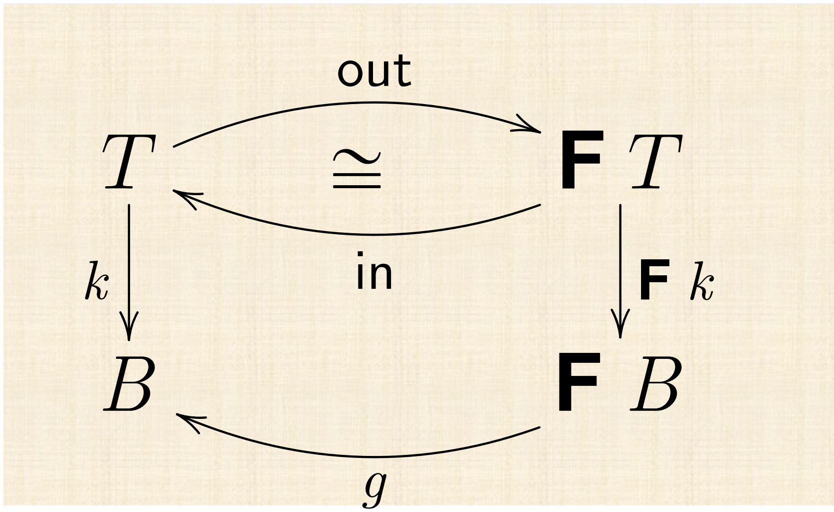
π_1
π_2
A
$A \times B$
B
f
$\langle f, g \rangle$
g
C



i_1
i_2
A
$A + B$
B
f
$[f, g]$
g
C

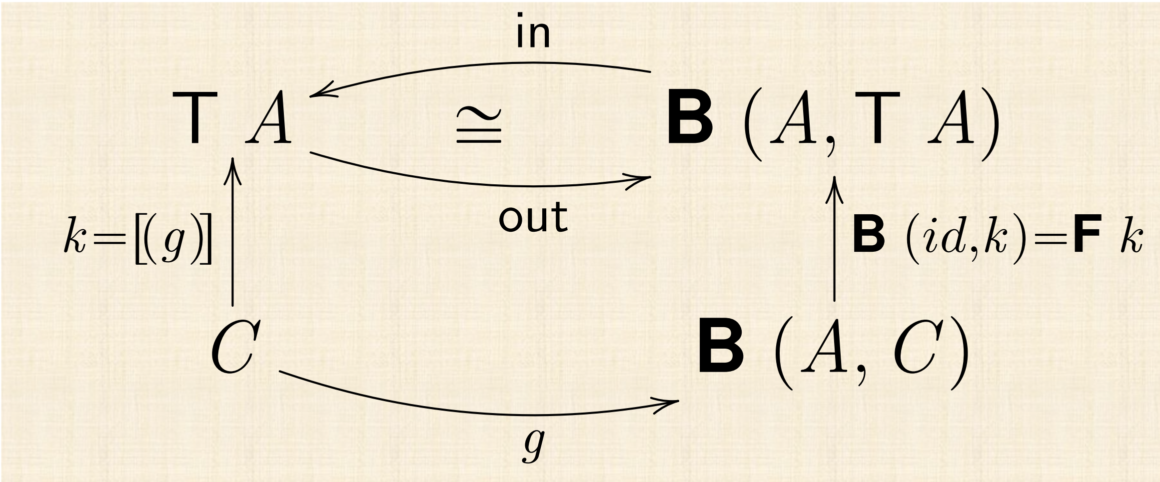


$\alpha\nu\alpha$ (ana)



$\kappa\alpha\tau\alpha$ (cata)

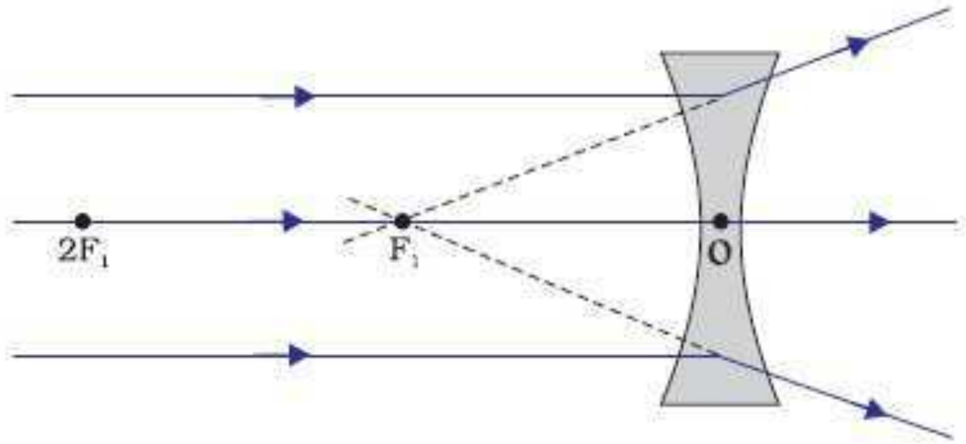
ANAMORFISMOS

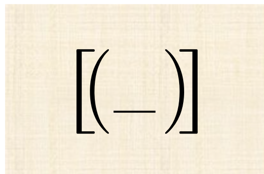


$$k = [\!(g)\!] \;\Leftrightarrow\; \mathsf{in} \cdot \mathbf{F}\, k \cdot g$$

ANAMORFISMOS

Universal-ana	$k = [\![g]\!] \quad \Leftrightarrow \quad \mathsf{out} \cdot k = (\mathsf{F}\, k) \cdot g$	(52)
Cancelamento-ana	$\mathsf{out} \cdot [\![g]\!] = \mathsf{F}\, [\![g]\!] \cdot g$	(53)
Reflexão-ana	$[\![\mathsf{out}]\!] = id_{\mathsf{T}}$	(54)
Fusão-ana	$[\![g]\!] \cdot f = [\![h]\!] \quad \Leftarrow \quad g \cdot f = (\mathsf{F}\, f) \cdot h$	(55)
Base-ana	$\mathsf{F}\, f \quad = \quad \mathsf{B}\, (id, f)$	(56)
Def-map-ana	$\mathsf{T}\, f \quad = \quad [\![\mathsf{B}(f, id) \cdot \mathsf{out}]\!]$	(57)
Absorção-ana	$\mathsf{T}\, f \cdot [\![g]\!] \quad = \quad [\![\mathsf{B}(f, id) \cdot g]\!]$	(58)


2F1
F1
O



ANAMORFISMOS

$$
\begin{array}{ccccc}
{\mathbb{N}_0}^* & & \xleftarrow{\quad\mathsf{in}\quad} & & 1 + \mathbb{N}_0 \times {\mathbb{N}_0}^* \\
\uparrow k & & & & \uparrow id+id\times k \\
\mathbb{N}_0 & \xrightarrow[\mathsf{out}_{\mathbb{N}_0}]{} & 1 + \mathbb{N}_0 & \xrightarrow[id+\langle \mathsf{succ}, id\rangle)]{} & 1 + \mathbb{N}_0 \times \mathbb{N}_0
\end{array}
$$

$$
\begin{array}{ll}
& k = [\!(\,(id + \langle \mathsf{succ}, id \rangle) \cdot \mathsf{out}_{\mathbb{N}_0}\,)\!] \\
\equiv & \qquad \{ \text{ ana-universal } \} \\
& k = \mathsf{in} \cdot (id + id \times k) \cdot (id + \langle \mathsf{succ}, id \rangle) \cdot \mathsf{out}_{\mathbb{N}_0} \\
\equiv & \qquad \{ \text{ isomorfismo } \mathsf{in}_{\mathbb{N}_0} \,/\, \mathsf{out}_{\mathbb{N}_0} \ \} \\
& k \cdot \mathsf{in}_{\mathbb{N}_0} = \mathsf{in} \cdot (id + id \times k) \cdot (id + \langle \mathsf{succ}, id \rangle) \\
\equiv & \qquad \{ \text{ functor-}+\text{; absorção-}\times \ \} \\
& k \cdot \mathsf{in}_{\mathbb{N}_0} = \mathsf{in} \cdot (id + \langle \mathsf{succ}, k \rangle)
\end{array}
$$

$\equiv$ { definições de in e $\mathsf{in}_{\mathbb{N}_0}$; fusão e absorção-$+$ }

$$[k \cdot \underline{0}, k \cdot \mathsf{succ}] = [\mathsf{nil}, \mathsf{cons} \cdot \langle \mathsf{succ}, k \rangle]$$

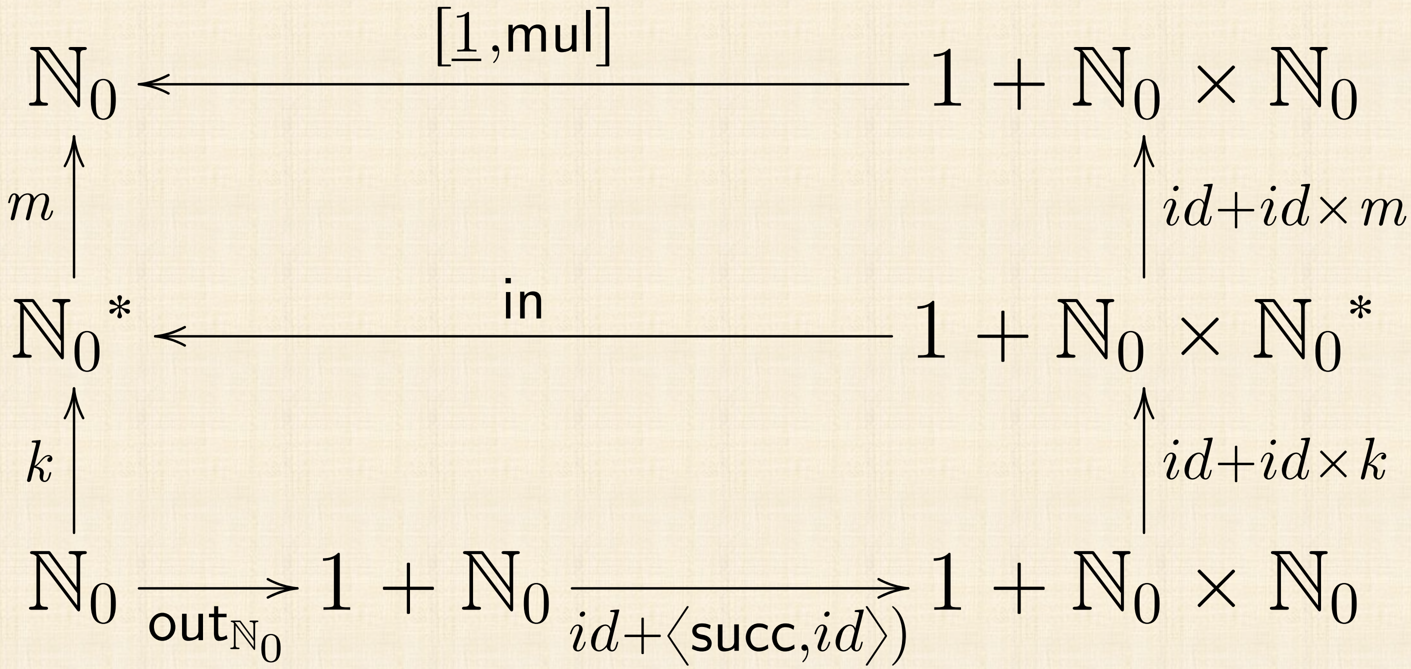
$\equiv$ { Eq-$+$ }

$$\begin{cases} k \cdot \underline{0} = \mathsf{nil} \\ k \cdot \mathsf{succ} = \mathsf{cons} \cdot \langle \mathsf{succ}, k \rangle \end{cases}$$

$\equiv$ { passagem a *pointwise* }

$$\begin{cases} k\ 0 = [\,] \\ k\ (n+1) = (n+1) : k\ n \end{cases}$$

$$
\begin{array}{ccccc}
{\mathbb{N}_0}^* & \xleftarrow{\quad\mathsf{in}\quad} & & & 1 + \mathbb{N}_0 \times {\mathbb{N}_0}^* \\
\uparrow{\scriptstyle k} & & & & \uparrow{\scriptstyle id+id\times k} \\
\mathbb{N}_0 & \xrightarrow[\mathsf{out}_{\mathbb{N}_0}]{} & 1 + \mathbb{N}_0 & \xrightarrow[id+\langle \mathsf{succ}, id\rangle)]{} & 1 + \mathbb{N}_0 \times \mathbb{N}_0
\end{array}
$$


$\mathbb{N}_0$
$[\underline{1}, \mathsf{mul}]$
$1 + \mathbb{N}_0 \times \mathbb{N}_0$
m
$id + id \times m$
$\mathbb{N}_0{}^*$
in
$1 + \mathbb{N}_0 \times \mathbb{N}_0{}^*$
k
$id + id \times k$
$\mathbb{N}_0$
$\mathsf{out}_{\mathbb{N}_0}$
$1 + \mathbb{N}_0$
$id + \langle \mathsf{succ}, id \rangle)$
$1 + \mathbb{N}_0 \times \mathbb{N}_0$

$$f = m \cdot k$$

$$\equiv \quad \{\ m = (\![\underline{1}, \mathsf{mul}]\!) \text{ e } k = [\!(\,(id + \langle \mathsf{succ}, id \rangle) \cdot \mathsf{out}_{\mathbb{N}_0}\,)\!]\ \}$$

$$f = (\![\underline{1}, \mathsf{mul}]\!) \cdot [\!(\,(id + \langle \mathsf{succ}, id \rangle) \cdot \mathsf{out}_{\mathbb{N}_0}\,)\!]$$

$$\equiv \quad \{\ \text{cancelamento-cata e cancelamento-ana}\ \}$$

$$f = [\underline{1}, \mathsf{mul}] \cdot \mathbf{F}\ m \cdot \mathsf{out} \cdot \mathsf{in} \cdot \mathbf{F}\ k \cdot (id + \langle \mathsf{succ}, id \rangle) \cdot \mathsf{out}_{\mathbb{N}_0}$$

$$\equiv \quad \{\ \mathsf{in} \cdot \mathsf{out} = id \text{ ; functor } \mathbf{F}\text{: } (\mathbf{F}\ m) \cdot (\mathbf{F}\ k) = \mathbf{F}\ (m \cdot k)\ \}$$

$$f = [\underline{1}, \mathsf{mul}] \cdot \mathbf{F}\ (m \cdot k) \cdot (id + \langle \mathsf{succ}, id \rangle) \cdot \mathsf{out}_{\mathbb{N}_0}$$

$$\equiv \quad \{\ \text{isomorfismo } \mathsf{in}_{\mathbb{N}_0} / \mathsf{out}_{\mathbb{N}_0} ;\ m \cdot k = f \text{ ; } \mathbf{F}\, f = id + id \times f\ \}$$

$$f \cdot \mathsf{in}_{\mathbb{N}_0} = [\underline{1}, \mathsf{mul}] \cdot (id + id \times f) \cdot (id + \langle \mathsf{succ}, id \rangle)$$

$$f \cdot \mathsf{in}_{\mathbb{N}_0} = [\underline{1}, \mathsf{mul}] \cdot (id + id \times f) \cdot (id + \langle \mathsf{succ}, id \rangle)$$

$\equiv$ { absorção-$+$; absorção-$\times$; etc }

$$f \cdot \mathsf{in}_{\mathbb{N}_0} = [\underline{1}, \mathsf{mul} \cdot \langle \mathsf{succ}, f \rangle]$$

$\equiv$ { Eq-$+$; $\mathsf{in}_{\mathbb{N}_0} = [\underline{0}, \mathsf{succ}]$ }

$$\begin{cases} f \cdot \underline{0} = \underline{1} \\ f \cdot \mathsf{succ} = \mathsf{mul} \cdot \langle \mathsf{succ}, f \rangle \end{cases}$$

$\equiv$ { introdução de variáveis }

$$\begin{cases} f\ 0 = 1 \\ f\ (n+1) = (n+1) \times f\ n \end{cases}$$

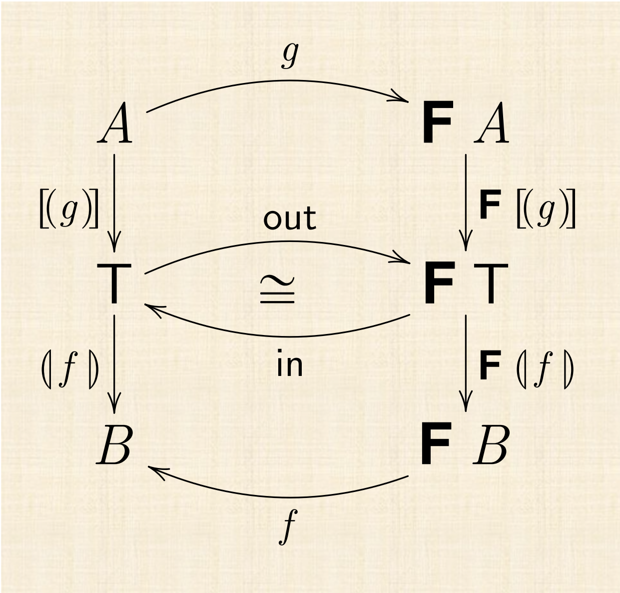
HILOMORFISMO

“Hylo + morphism”

ξύλο = matéria, coisa

$$[\![f, g]\!] = (\!| f |\!) \cdot [\!(g)\!]$$

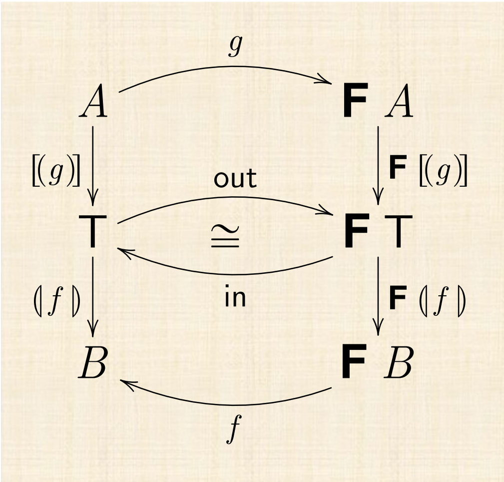
HILOMORFISMO

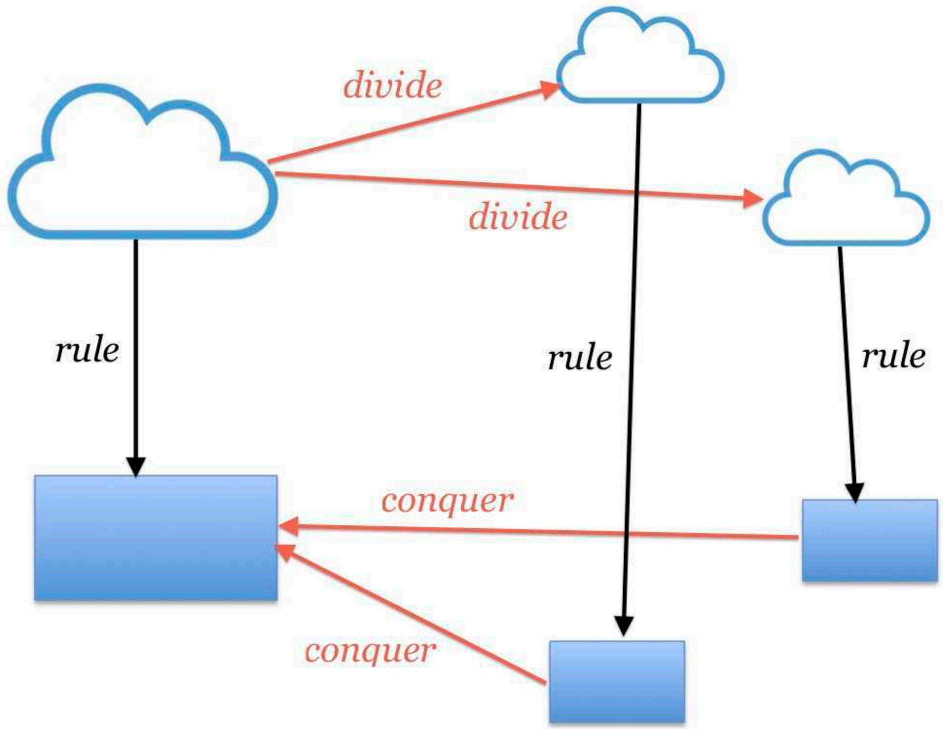


$$[\![f, g]\!] = (\!| f |\!) \cdot [\!(g)\!]$$

Cálculo de Programas
Aula T10

'DIVIDE & CONQUER'





`DIVIDE & CONQUER’

$$C \xleftarrow{\;(\!|\, - \,|\!)\;} \mathsf{T} \xleftarrow{\;[\!(\, - \,)\!]\;} A$$

OS ALGORITMOS E A OPINIÃO PÚBLICA

INTERNET

Autoridade da Concorrência avisa que algoritmos de preços podem infringir a lei

Karla Pequenino

02 de Julho de 2019

43

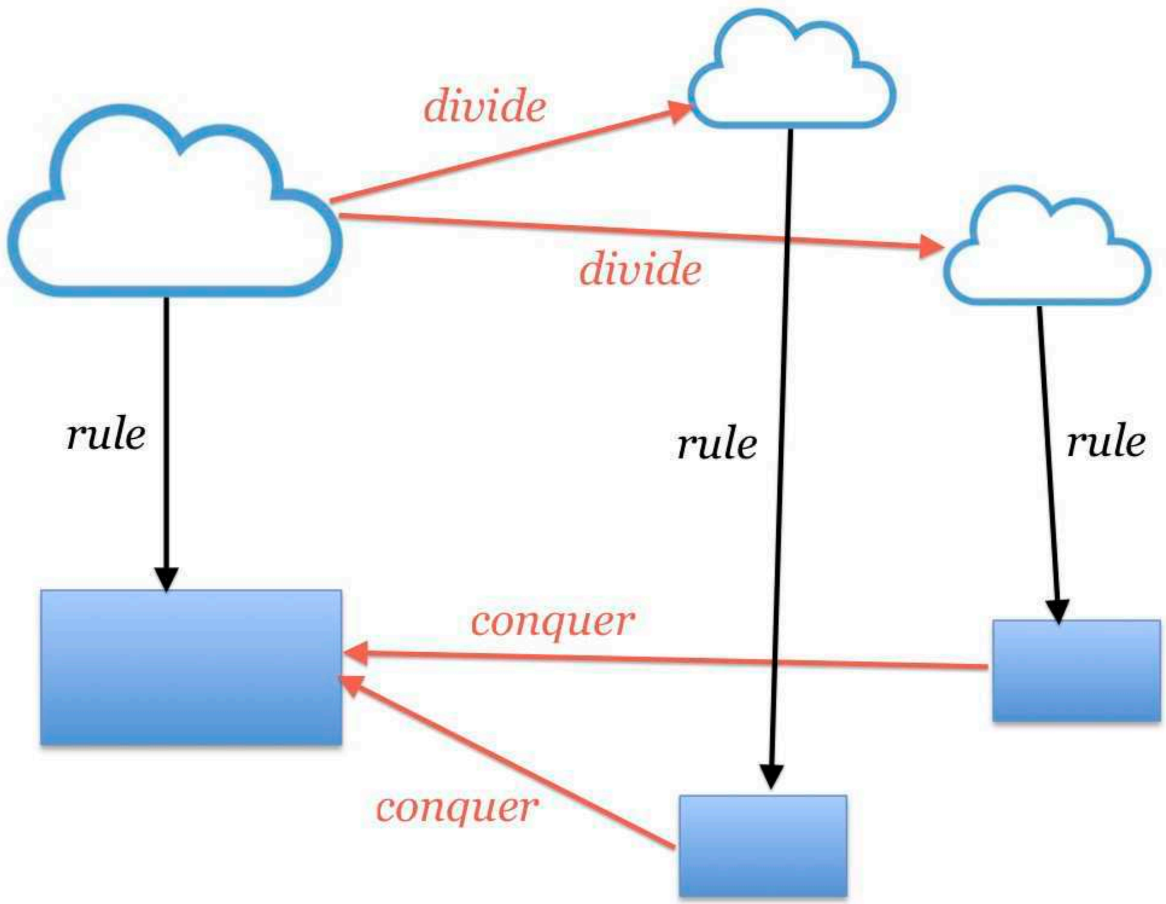
ALGORÍTMO = HILOMORFISMO

“Hylomorphism” = “divide & conquer”

ξύλο = matéria, coisa

$$[\![f, g]\!] = (\!| f |\!) \cdot [\!(g)\!]$$

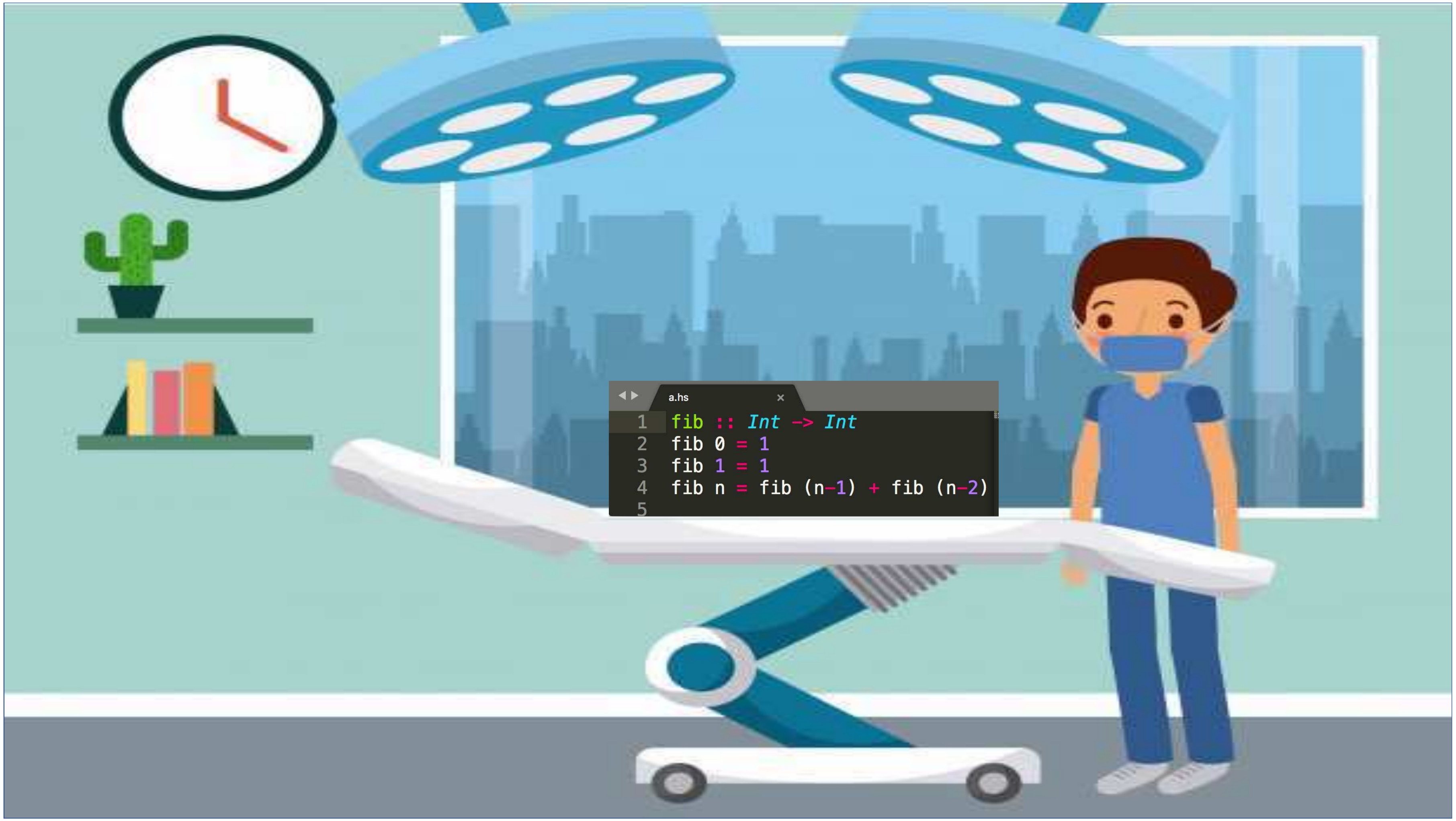
`DIVIDE & CONQUER'

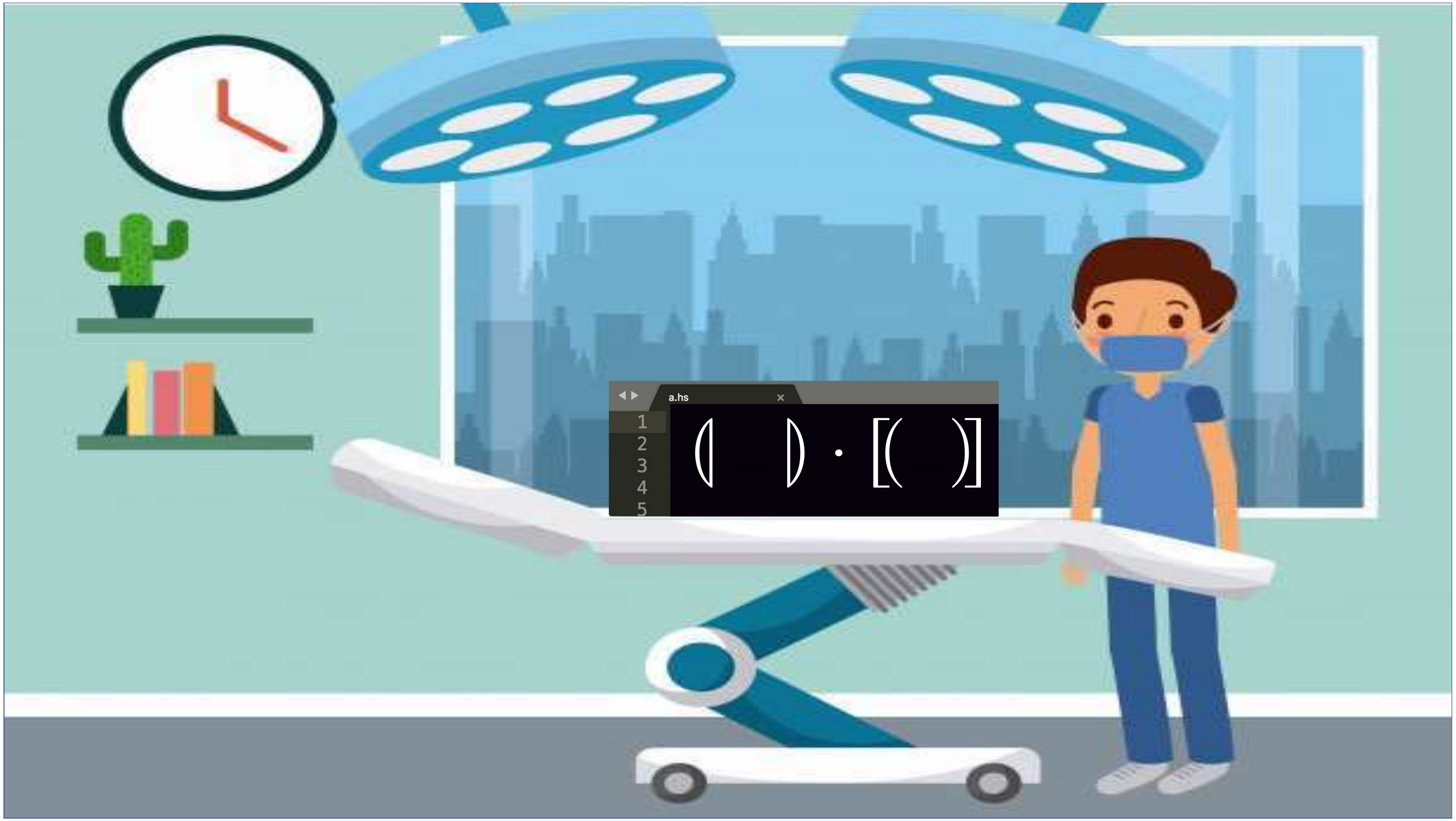


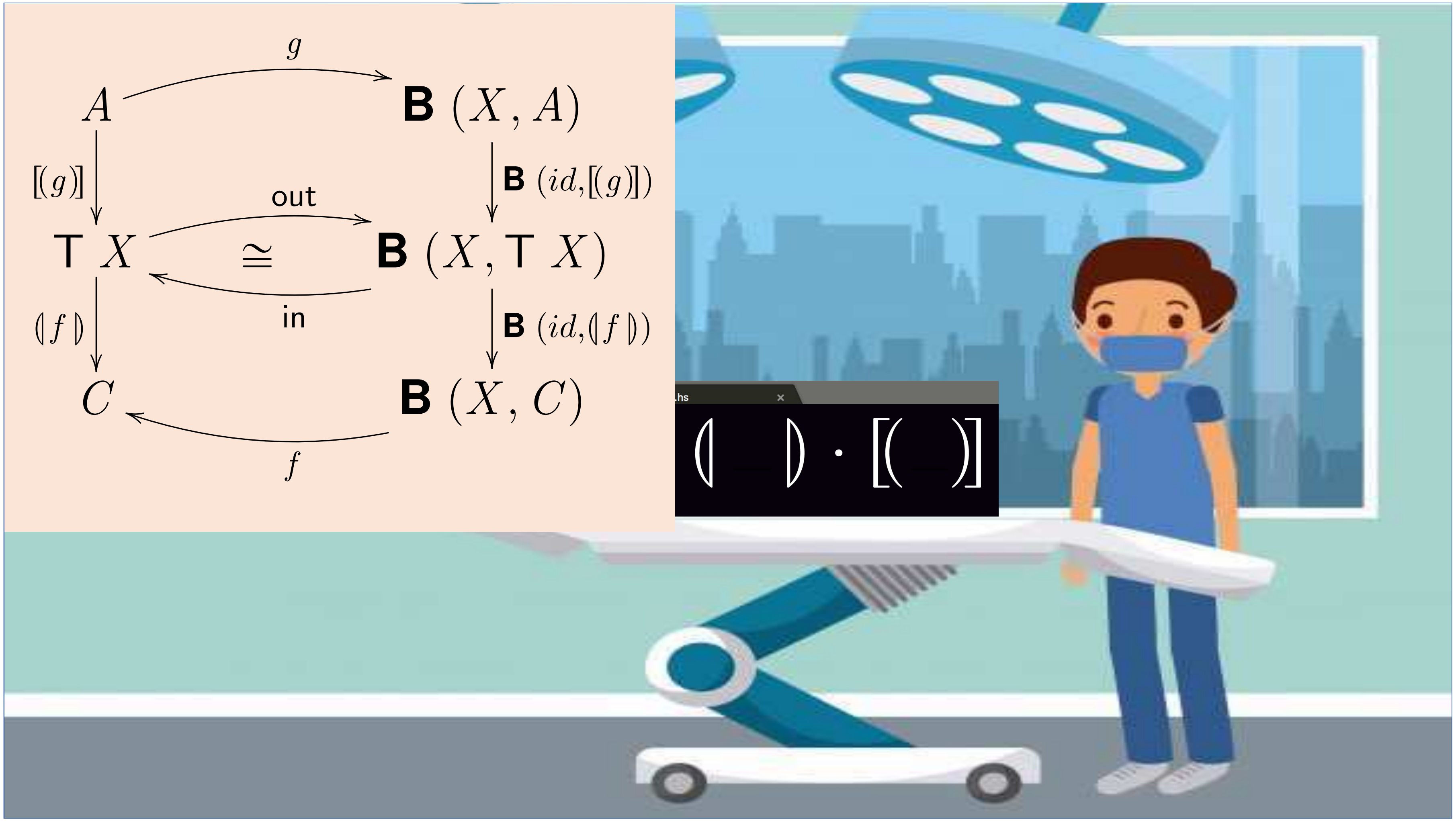
Alguns muito bons a **dividir**...

Tortuous Convolvulus
(*Asterix and the Roman Agent*, by Goscinny & Uderzo, Hachette Livre, 1970)

...outros (quase!) tão bons a **conquistar**:


a.hs
fib :: Int -> Int
fib 0 = 1
fib 1 = 1
fib n = fib (n-1) + fib (n-2)

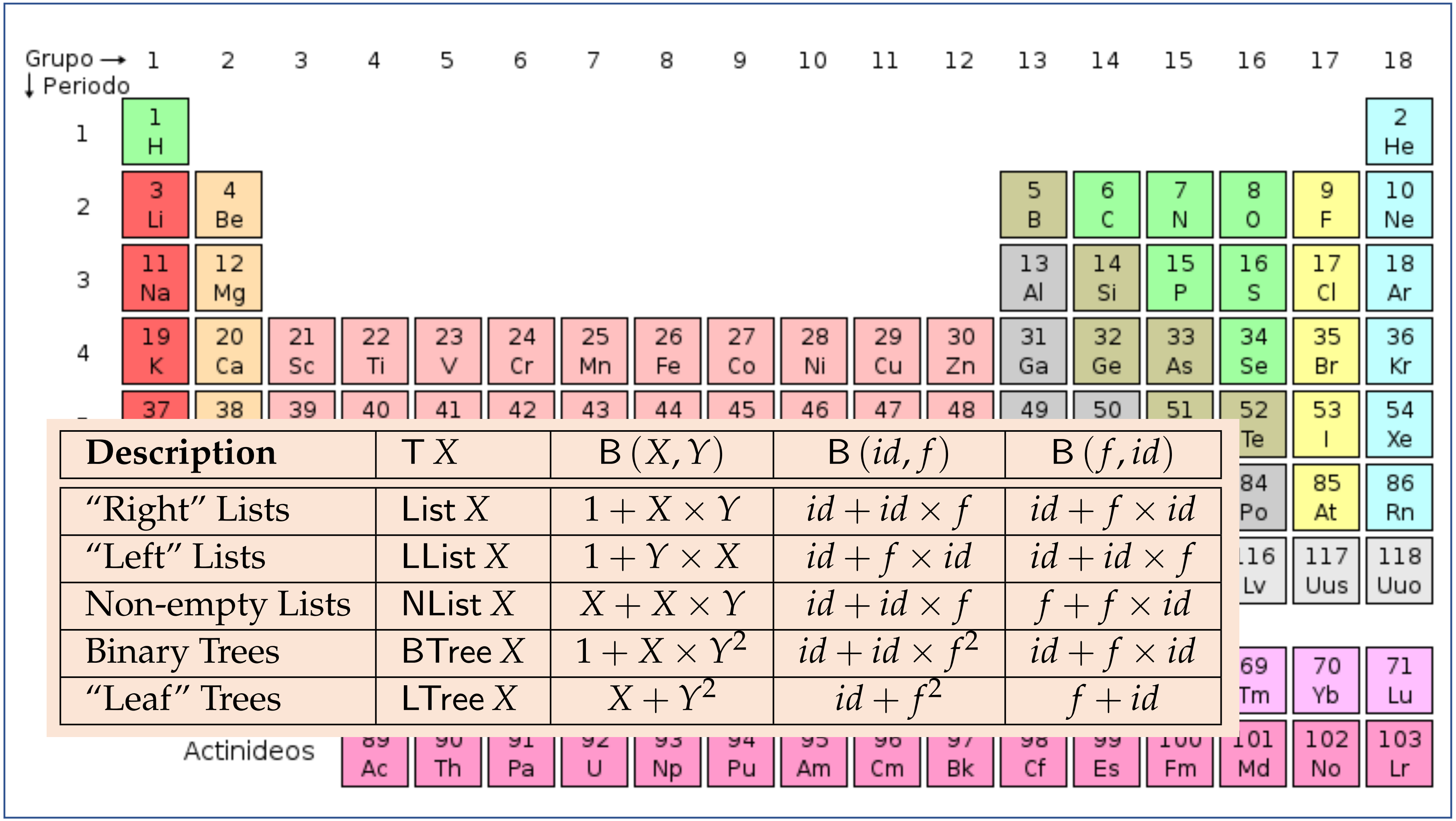

a.hs



g
A
B (X, A)
[(g)]
B (id,[(g)])
out
T X
≅
B (X, T X)
in
(|f|)
B (id,(|f|))
C
B (X, C)
f
.hs
(| |) · [( )]

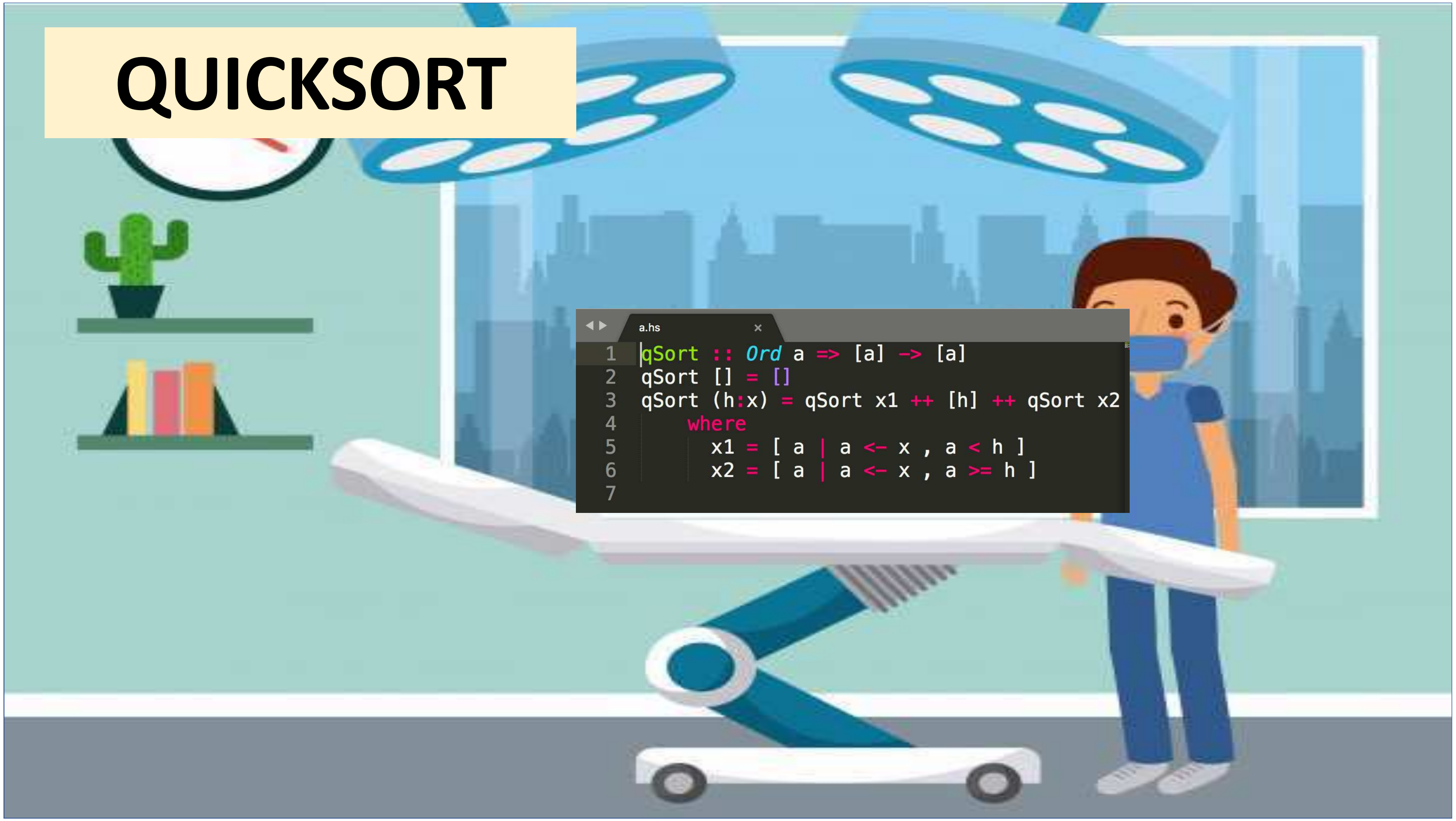
"ARCA ALGORÍTMICA"

Grupo → / ↓ Periodo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	1 H																	2 He
2	3 Li	4 Be											5 B	6 C	7 N	8 O	9 F	10 Ne
3	11 Na	12 Mg											13 Al	14 Si	15 P	16 S	17 Cl	18 Ar
4	19 K	20 Ca	21 Sc	22 Ti	23 V	24 Cr	25 Mn	26 Fe	27 Co	28 Ni	29 Cu	30 Zn	31 Ga	32 Ge	33 As	34 Se	35 Br	36 Kr
5	37 Rb	38 Sr	39 Y	40 Zr	41 Nb	42 Mo	43 Tc	44 Ru	45 Rh	46 Pd	47 Ag	48 Cd	49 In	50 Sn	51 Sb	52 Te	53 I	54 Xe
6	55 Cs	56 Ba		72 Hf	73 Ta	74 W	75 Re	76 Os	77 Ir	78 Pt	79 Au	80 Hg	81 Tl	82 Pb	83 Bi	84 Po	85 At	86 Rn
7	87 Fr	88 Ra		104 Rf	105 Db	106 Sg	107 Bh	108 Hs	109 Mt	110 Ds	111 Rg	112 Cn	113 Uut	114 Fl	115 Uup	116 Lv	117 Uus	118 Uuo
Lantanideos				57 La	58 Ce	59 Pr	60 Nd	61 Pm	62 Sm	63 Eu	64 Gd	65 Tb	66 Dy	67 Ho	68 Er	69 Tm	70 Yb	71 Lu
Actinideos				89 Ac	90 Th	91 Pa	92 U	93 Np	94 Pu	95 Am	96 Cm	97 Bk	98 Cf	99 Es	100 Fm	101 Md	102 No	103 Lr

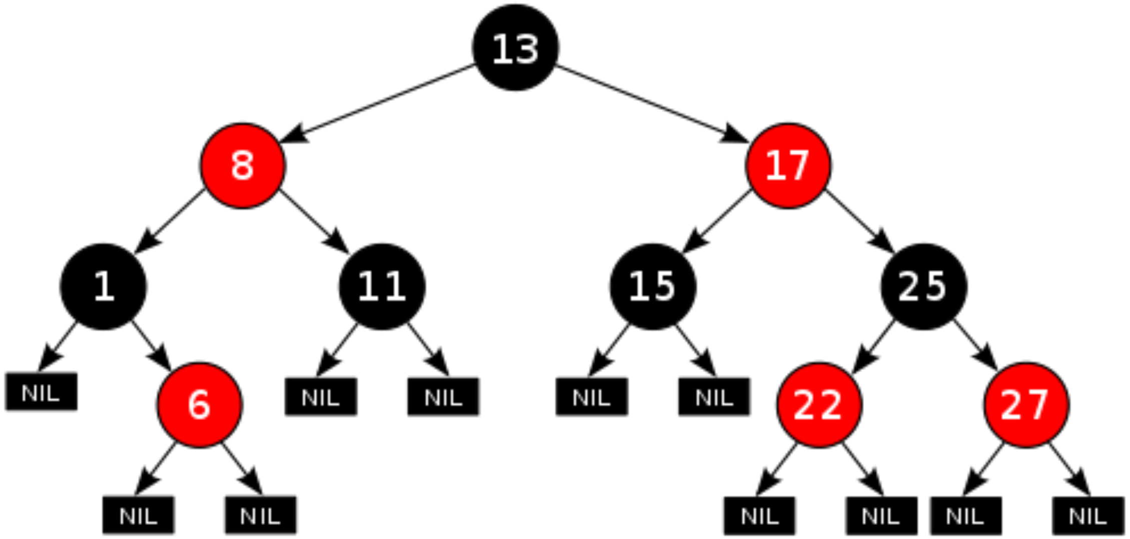


Description	T X	B (X, Y)	B (id, f)	B (f, id)
"Right" Lists	List X	$1 + X \times Y$	$id + id \times f$	$id + f \times id$
"Left" Lists	LList X	$1 + Y \times X$	$id + f \times id$	$id + id \times f$
Non-empty Lists	NList X	$X + X \times Y$	$id + id \times f$	$f + f \times id$
Binary Trees	BTree X	$1 + X \times Y^2$	$id + id \times f^2$	$id + f \times id$
"Leaf" Trees	LTree X	$X + Y^2$	$id + f^2$	$f + id$

QUICKSORT



```haskell
qSort :: Ord a => [a] -> [a]
qSort [] = []
qSort (h:x) = qSort x1 ++ [h] ++ qSort x2
    where
      x1 = [ a | a <- x , a < h ]
      x2 = [ a | a <- x , a >= h ]
```

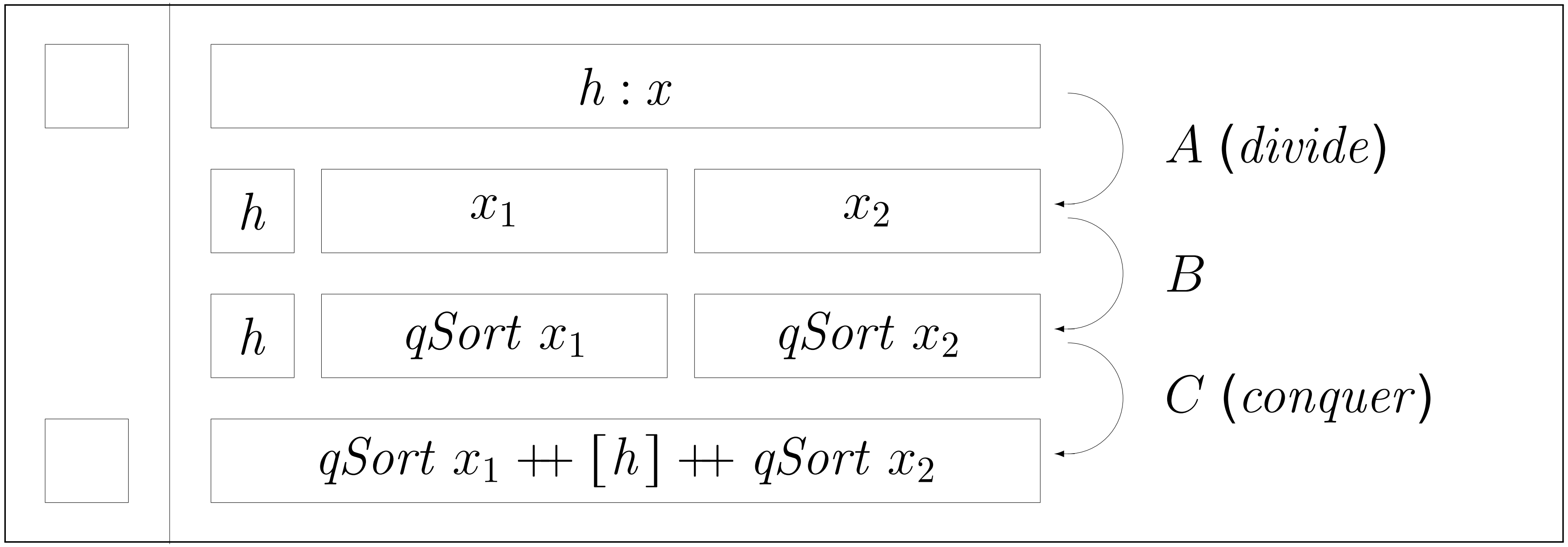



```haskell
data BTree a = Empty | Node (a, (BTree a, BTree a))
```

QUICKSORT

a.hs

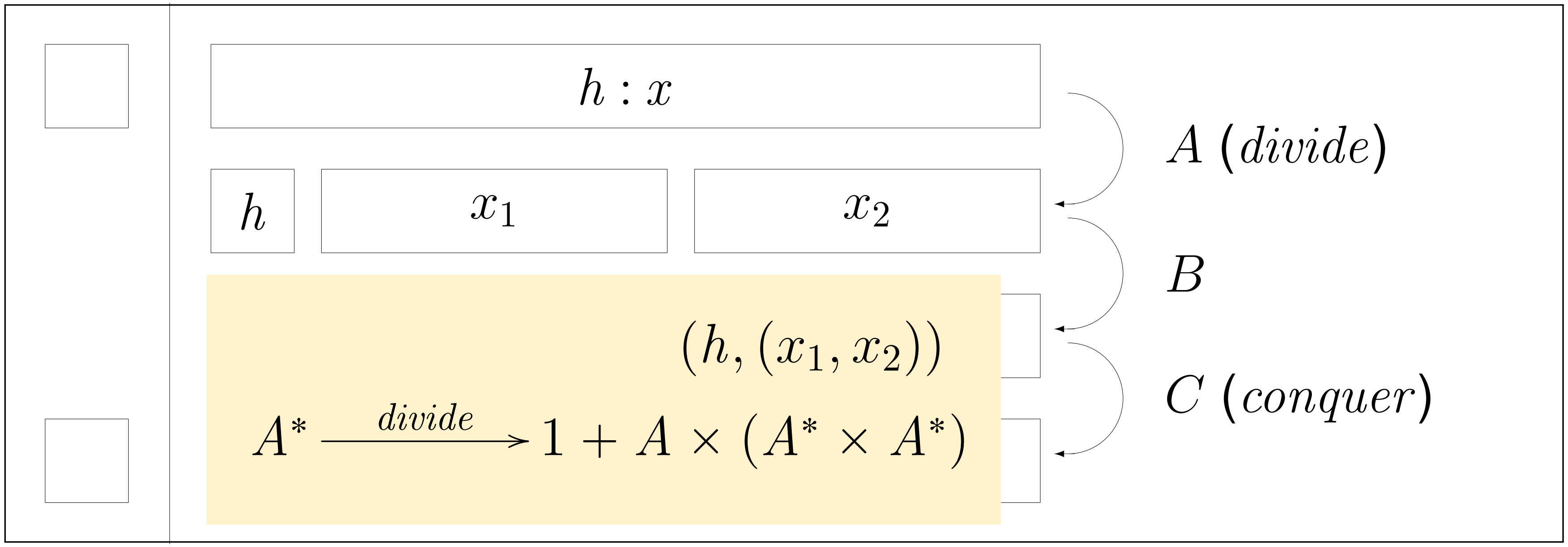
```haskell
qSort :: Ord a => [a] -> [a]
qSort [] = []
qSort (h:x) = qSort x1 ++ [h] ++ qSort x2
    where
      x1 = [ a | a <- x , a < h ]
      x2 = [ a | a <- x , a >= h ]
```



QUICKSORT

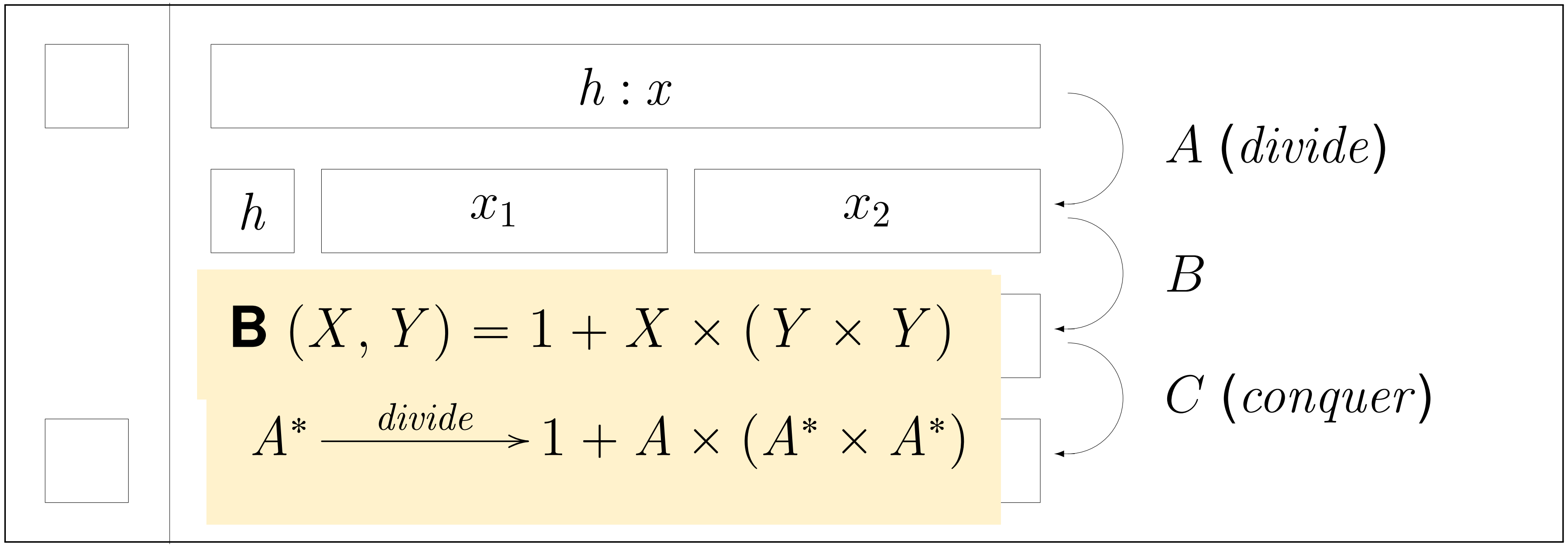
a.hs

```haskell
qSort :: Ord a => [a] -> [a]
qSort [] = []
qSort (h:x) = qSort x1 ++ [h] ++ qSort x2
    where
      x1 = [ a | a <- x , a < h ]
      x2 = [ a | a <- x , a >= h ]
```



QUICKSORT

```haskell
qSort :: Ord a => [a] -> [a]
qSort [] = []
qSort (h:x) = qSort x1 ++ [h] ++ qSort x2
    where
        x1 = [ a | a <- x , a < h ]
        x2 = [ a | a <- x , a >= h ]
```



QUICKSORT

```haskell
qSort :: Ord a => [a] -> [a]
qSort [] = []
qSort (h:x) = qSort x1 ++ [h] ++ qSort x2
    where
      x1 = [ a | a <- x , a < h ]
      x2 = [ a | a <- x , a >= h ]
```

$$\begin{cases} qSort \cdot \mathsf{nil} = \mathsf{nil} \\ qSort \cdot \mathsf{cons} = f_2 \cdot (id \times (qSort \times qSort)) \cdot g_2 \end{cases}$$

$$f_2\ (h, (y_1, y_2)) = y_1 +\!\!+ [h] +\!\!+ y_2$$

$$\begin{array}{l} g_2\ (h, x) = (h, (x_1, x_2)) \\ \quad \textbf{where} \\ \qquad x_1 = [a \mid a \leftarrow x, a < h] \\ \qquad x_2 = [a \mid a \leftarrow x, a \geqslant h] \end{array}$$

QUICKSORT

$$\mathsf{B}\,(X, Y) = 1 + X \times (Y \times Y)$$

$$\begin{cases} qSort \cdot \mathsf{nil} = \mathsf{nil} \\ qSort \cdot \mathsf{cons} = f_2 \cdot (id \times (qSort \times qSort)) \cdot g_2 \end{cases}$$

$$\equiv \qquad \{ \text{ fusão-+, absorção-+, eq-+ etc } \}$$

$$qSort \cdot \mathsf{in} = [\mathsf{nil}, f_2] \cdot (id + id \times qSort^2) \cdot (id + g_2)$$

$$\equiv \qquad \{ \text{ isomorfismo in / out } \}$$

$$qSort = \underbrace{[\mathsf{nil}, f_2]}_{conquer} \cdot \underbrace{(id + id \times qSort^2}_{\mathsf{B}\,(id, qSort)} \cdot \underbrace{(id + g_2)}_{divide} \cdot \mathsf{out}$$

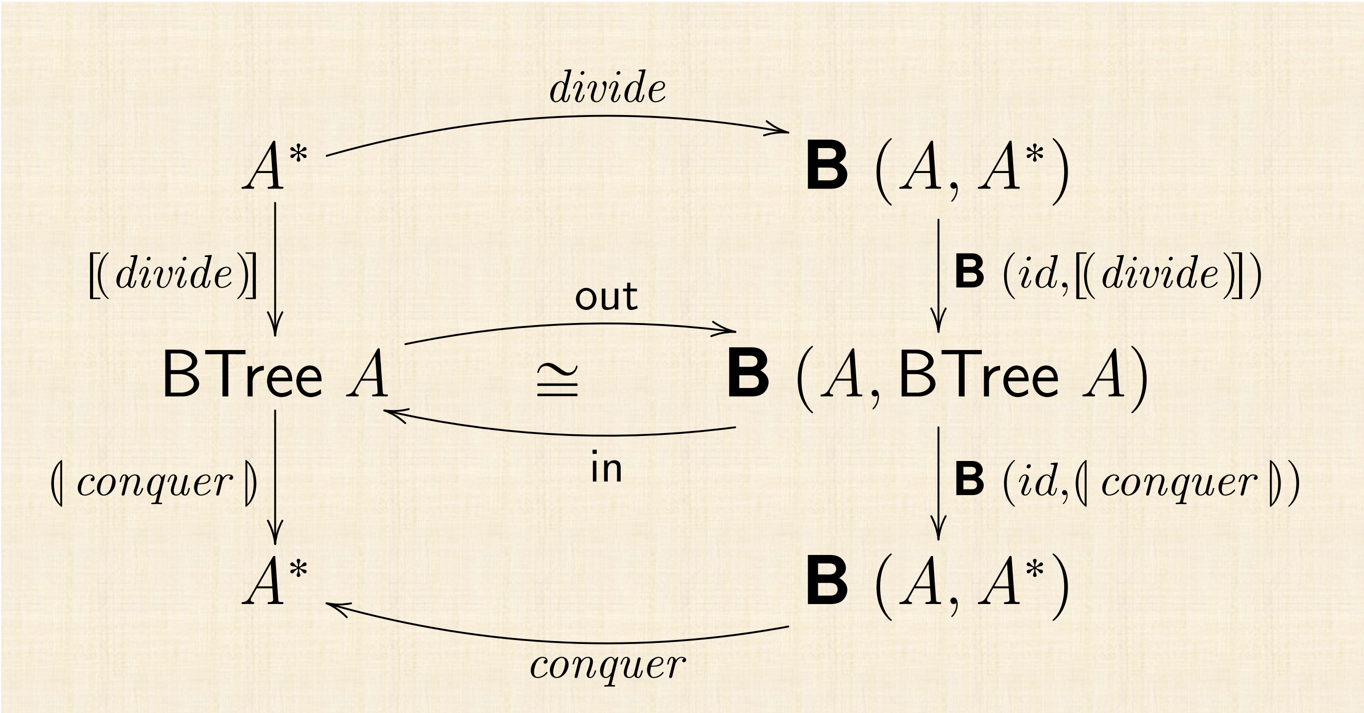
QUICKSORT

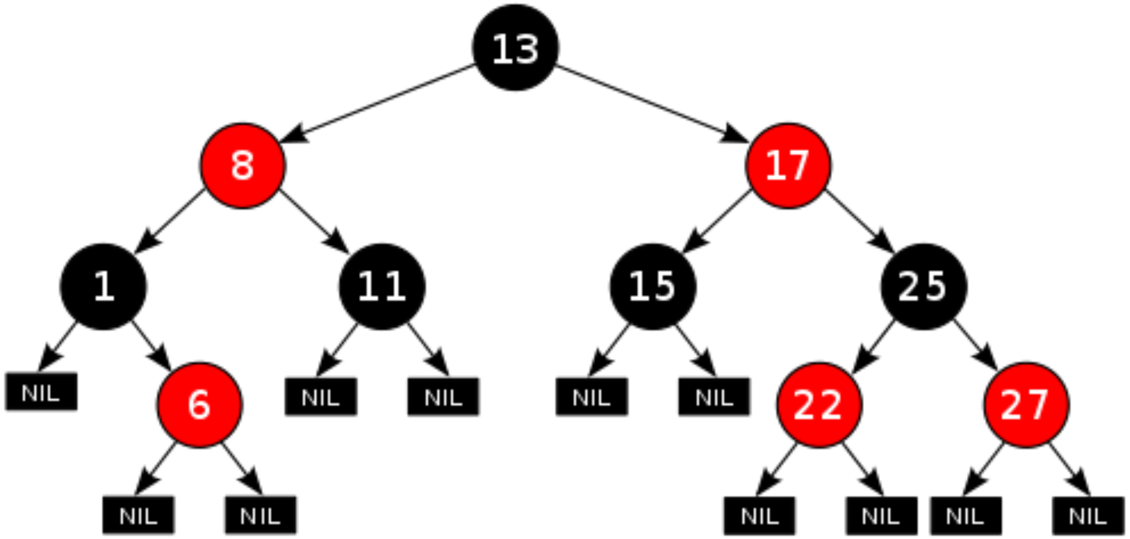
$$\mathbf{B}\,(X, Y) = 1 + X \times (Y \times Y)$$

Description	T X	B (X, Y)	B (id, f)	B (f, id)
"Right" Lists	List X	$1 + X \times Y$	$id + id \times f$	$id + f \times id$
"Left" Lists	LList X	$1 + Y \times X$	$id + f \times id$	$id + id \times f$
Non-empty Lists	NList X	[illegible] $+ X \times Y$	$id + id \times f$	$f + f \times id$
Binary Trees	BTree X	$1 + X \times Y^2$	$id + id \times f^2$	$id + f \times id$
"Leaf" Trees	LTree X	$X + Y^2$	$id + f^2$	$f + id$

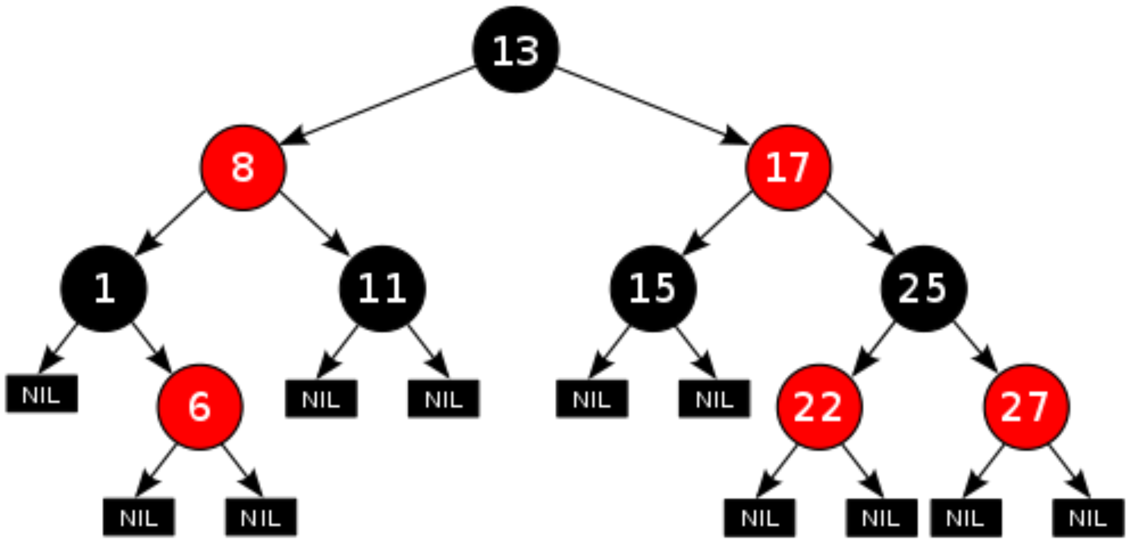
QUICKSORT

```haskell
qSort :: Ord a => [a] -> [a]
qSort [] = []
qSort (h:x) = qSort x1 ++ [h] ++ qSort x2
    where
      x1 = [ a | a <- x , a < h ]
      x2 = [ a | a <- x , a >= h ]
```



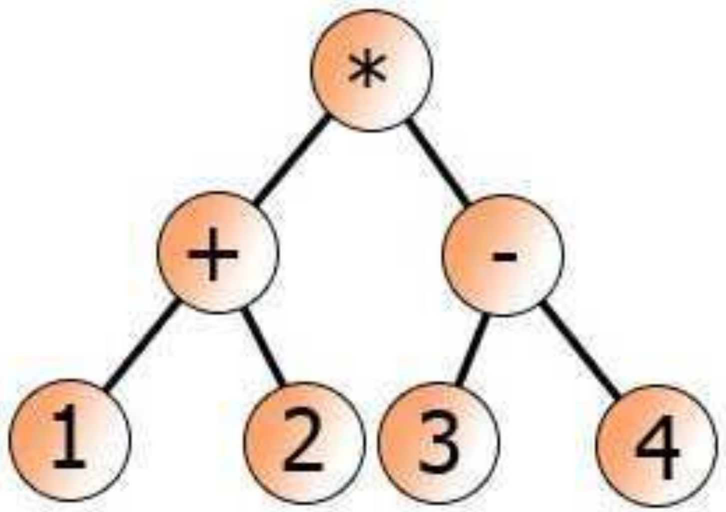



```
t = anaB divide [13,8,17,1,6,11,25,15,27,22]

data BTree a = Empty | Node (a, (BTree a, BTree a))
```

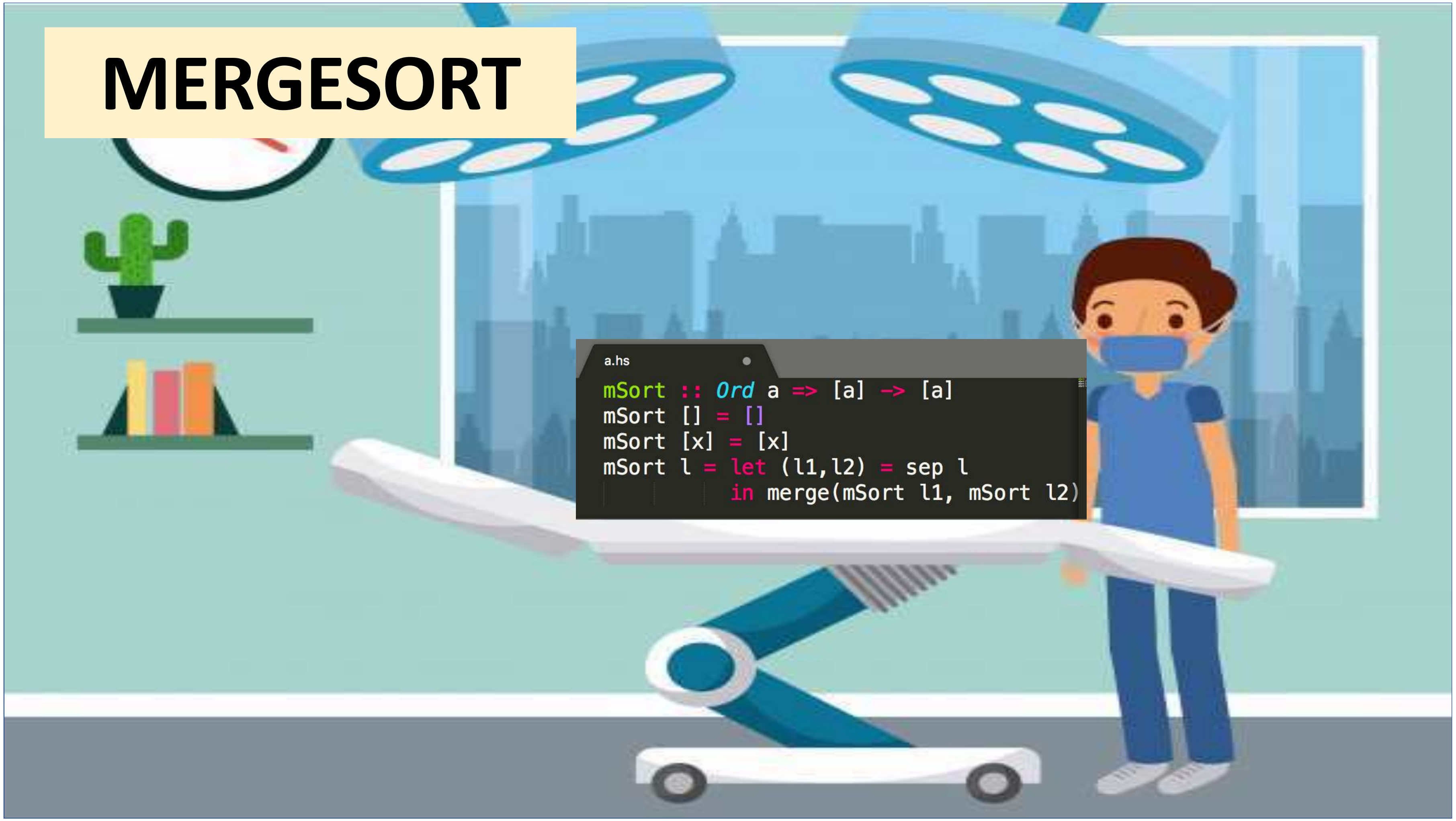



```
*Cp> anaB divide [13,8,17,1,6,11,25,15,27,22]
Node (13,(Node (8,(Node (1,(Empty,Node (6,(Empty,Empty)))),Node
(11,(Empty,Empty)))),Node (17,(Node (15,(Empty,Empty)),Node (25,
(Node (22,(Empty,Empty)),Node (27,(Empty,Empty))))))))
```

TRAVESSIAS

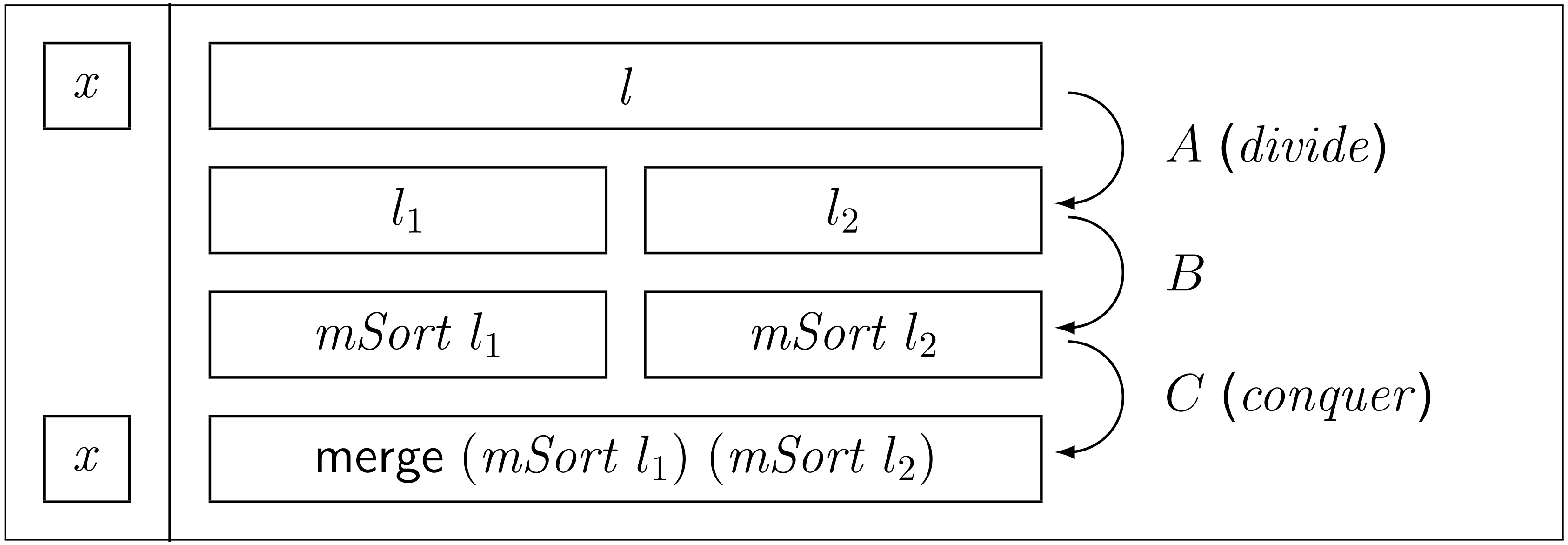



```
e = Node ("*",(
        Node ("+",(
           Node ("1",(Empty,Empty)),
           Node ("2",(Empty,Empty)))),
        Node ("-",(
           Node ("3",(Empty,Empty)),
           Node ("4",(Empty,Empty))))
        ))
```

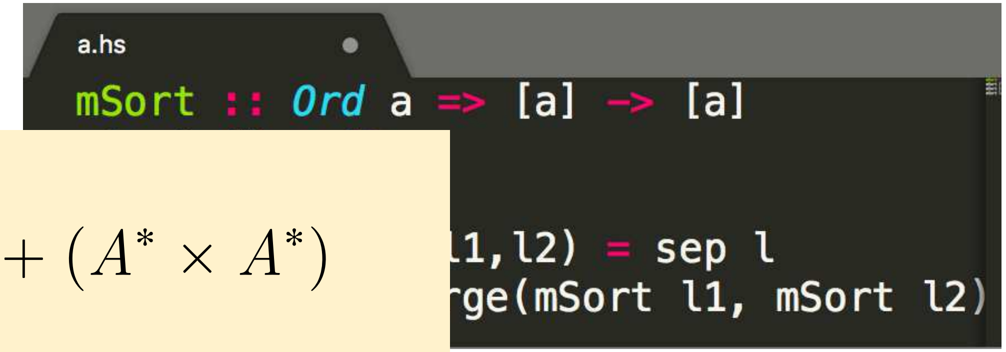

MERGESORT
a.hs
mSort :: Ord a => [a] -> [a]
mSort [] = []
mSort [x] = [x]
mSort l = let (l1,l2) = sep l
in merge(mSort l1, mSort l2)

MERGESORT

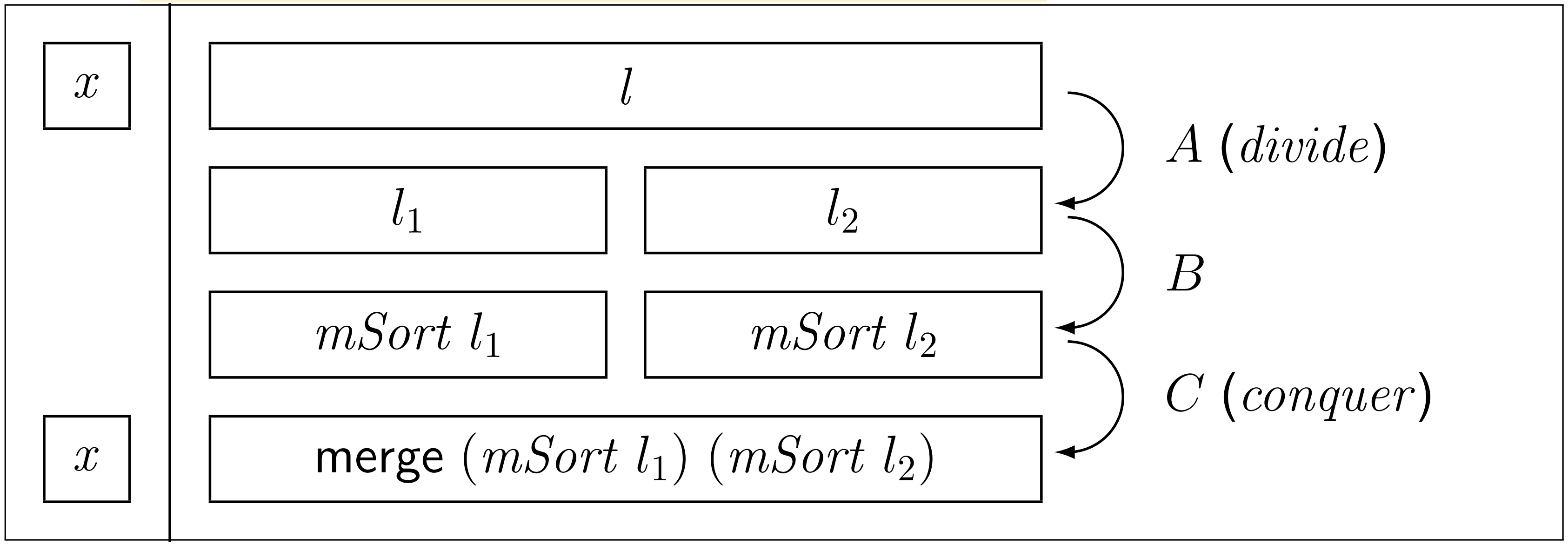
```haskell
mSort :: Ord a => [a] -> [a]
mSort [] = []
mSort [x] = [x]
mSort l = let (l1,l2) = sep l
          in merge(mSort l1, mSort l2)
```



MERGESORT



$$divide : A^* \to A + (A^* \times A^*)$$

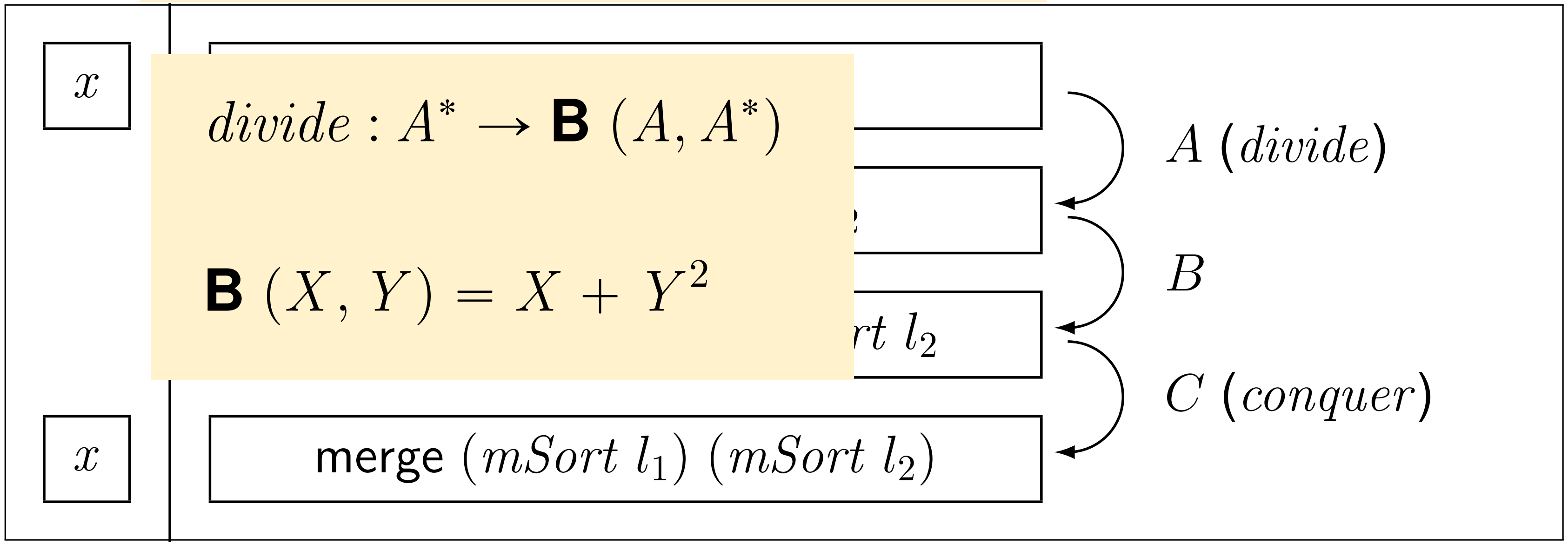


MERGESORT

a.hs

```
mSort :: Ord a => [a] -> [a]
l1,l2) = sep l
rge(mSort l1, mSort l2)
```

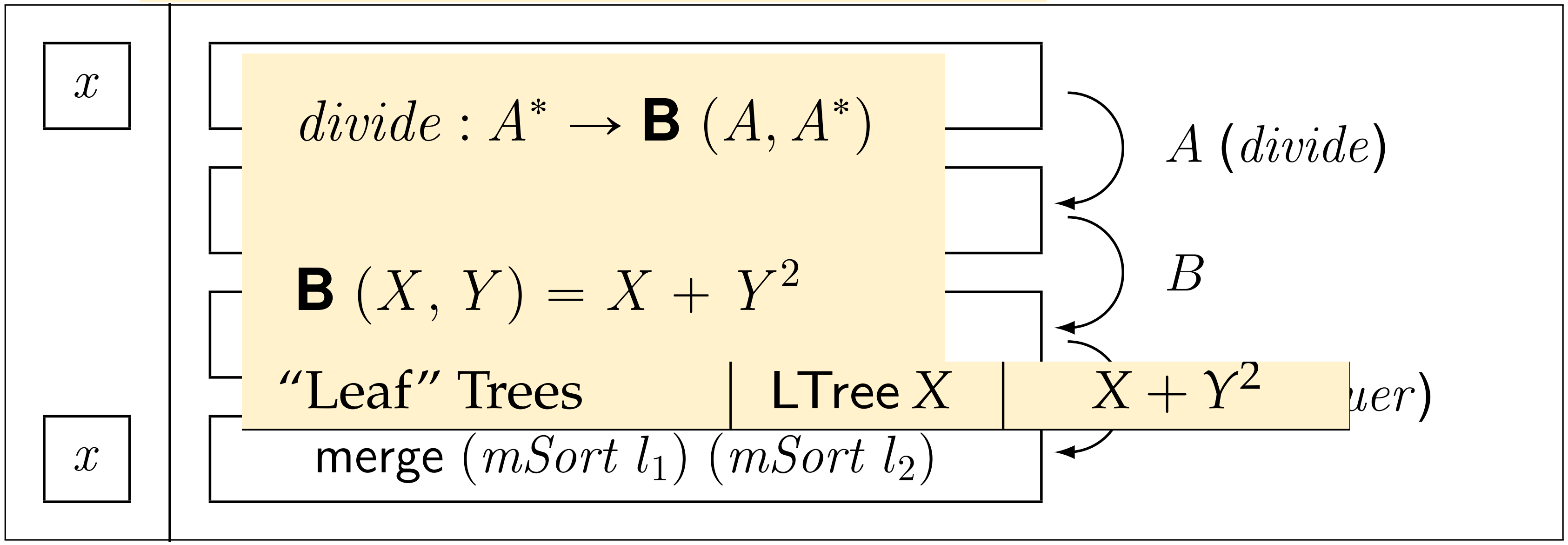
$$divide : A^* \rightarrow A + (A^* \times A^*)$$



MERGESORT

```
a.hs
mSort :: Ord a => [a] -> [a]
...
...l1,l2) = sep l
...rge(mSort l1, mSort l2)
```

$$divide : A^* \to A + (A^* \times A^*)$$

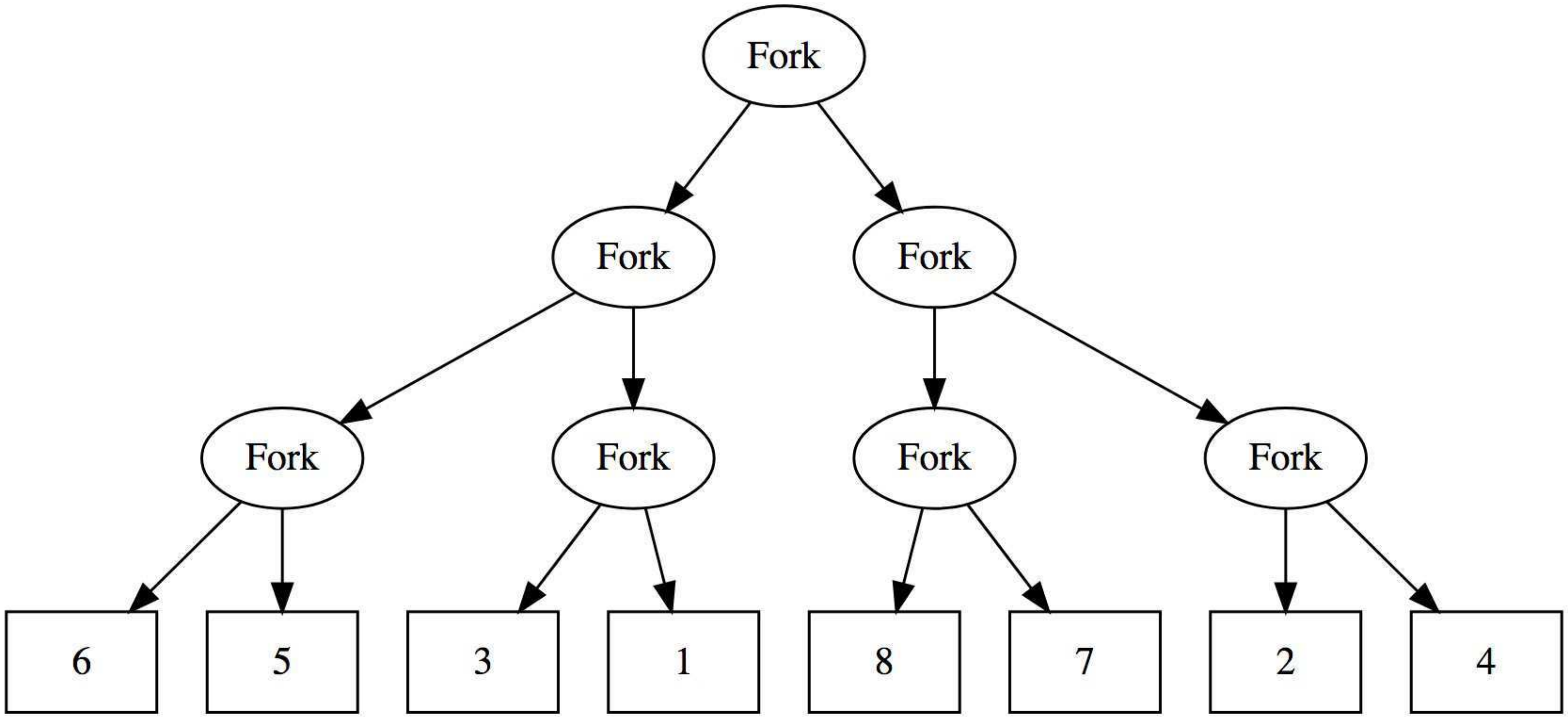


$$divide : A^* \to \mathbf{B}\ (A, A^*)$$

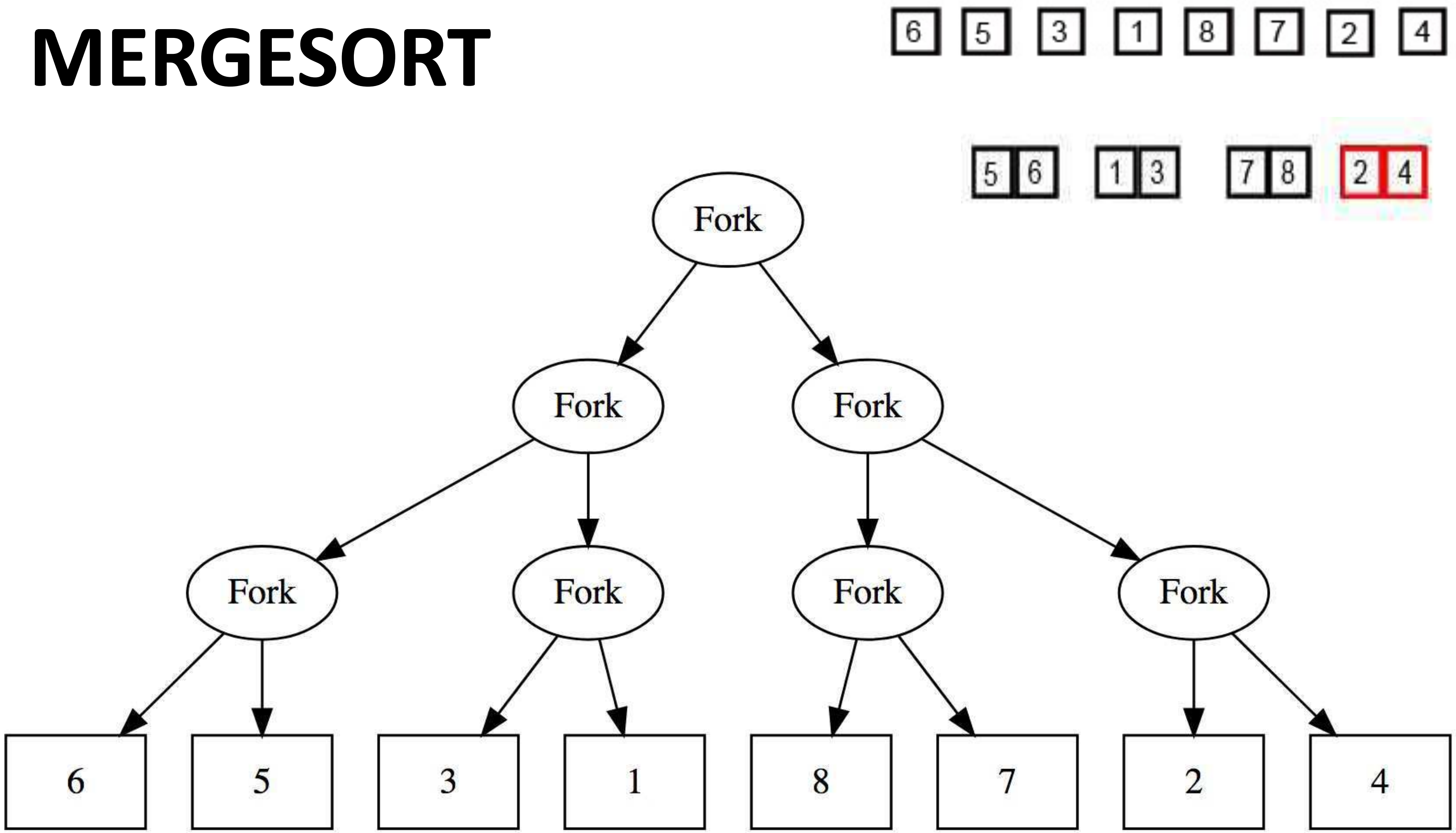
$$\mathbf{B}\ (X, Y) = X + Y^2$$

"Leaf" Trees	LTree X	$X + Y^2$

MERGESORT



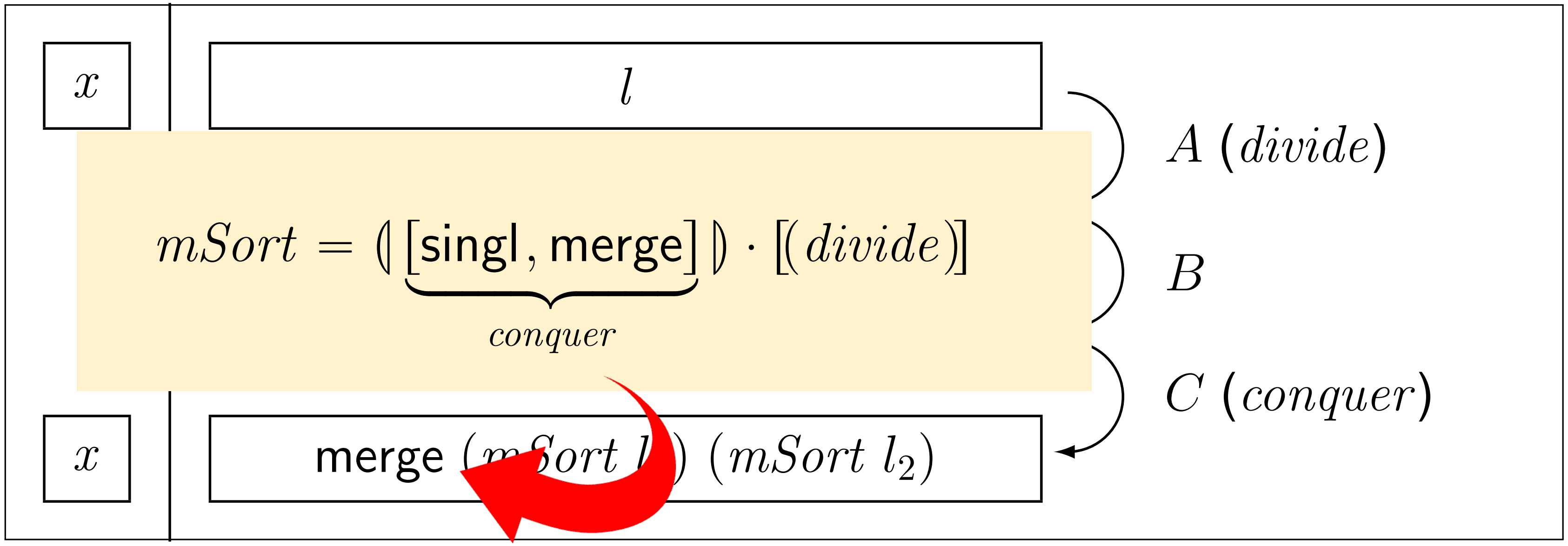
MERGESORT



6 5 3 1 8 7 2 4
5 6 1 3 7 8 2 4
Fork
Fork
Fork
Fork
Fork
Fork
Fork
6
5
3
1
8
7
2
4

MERGESORT

```haskell
mSort :: Ord a => [a] -> [a]
mSort [] = []
mSort [x] = [x]
mSort l = let (l1,l2) = sep l
              in merge(mSort l1, mSort l2)
```



$$mSort = (\!|\underbrace{[\mathsf{singl}, \mathsf{merge}]}_{conquer}|\!) \cdot [\!(divide)\!]$$

work — ghc -B/Library/Frameworks/GHC.framework/Versions/8.2.2-x86_64/usr/lib/ghc-8.2.2 --interactive Cp — 80×21

~/work — vi a.hs | ...ramework/Versions/8.2.2-x86_64/usr/lib/ghc-8.2.2 --interactive Cp

```
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp> :{
*Cp|
*Cp| divide (l,[])                    = i1 l
*Cp| divide ([],r)                    = i1 r
*Cp| divide (x:xs,y:ys) | x < y       = i2 (x,(xs,y:ys))
*Cp|                    | otherwise   = i2 (y,(x:xs,ys))
*Cp|
*Cp| :}
*Cp>
*Cp> :t divide
divide :: Ord a => ([a], [a]) -> Either [a] (a, ([a], [a]))
*Cp>
```

work — ghc -B/Library/Frameworks/GHC.framework/Versions/8.2.2-x86_64/usr/lib/ghc-8.2.2 --interactive Cp — 80×21

~/work — vi a.hs | ...ramework/Versions/8.2.2-x86_64/usr/lib/ghc-8.2.2 --interactive Cp

$$divide : A^* \times A^* \rightarrow A^* + A \times (A^* \times A^*)$$

```
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp> :{
*Cp|
*Cp| divide (l,[])                    = i1 l
*Cp| divide ([],r)                    = i1 r
*Cp| divide (x:xs,y:ys) | x < y       = i2 (x,(xs,y:ys))
*Cp|                    | otherwise = i2 (y,(x:xs,ys))
*Cp|
*Cp| :}
*Cp>
*Cp> :t divide
divide :: Ord a => ([a], [a]) -> Either [a] (a, ([a], [a]))
*Cp>
```

work — ghc -B/Library/Frameworks/GHC.framework/Versions/8.2.2-x86_64/usr/lib/ghc-8.2.2 --interactive Cp — 80×21

~/work — vi a.hs | ...ramework/Versions/8.2.2-x86_64/usr/lib/ghc-8.2.2 --interactive Cp

$$divide : A^* \times A^* \rightarrow A^* + A \times (A^* \times A^*)$$

$$divide : A^* \times A^* \rightarrow \mathbf{B}\,(A^*, A, (A^* \times A^*))$$

```
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp> :{
*Cp|
*Cp| divide (l,[])                          = i1 l
*Cp| divide ([],r)                          = i1 r
*Cp| divide (x:xs,y:ys) | x < y      = i2 (x,(xs,y:ys))
*Cp|                    | otherwise = i2 (y,(x:xs,ys))
*Cp|
*Cp| :}
*Cp>
*Cp> :t divide
divide :: Ord a => ([a], [a]) -> Either [a] (a, ([a], [a]))
*Cp>
```

work — ghc -B/Library/Frameworks/GHC.framework/Versions/8.2.2-x86_64/usr/lib/ghc-8.2.2 --interactive Cp — 80×21

~/work — vi a.hs | ...ramework/Versions/8.2.2-x86_64/usr/lib/ghc-8.2.2 --interactive Cp

$$divide : A^* \times A^* \rightarrow A^* + A \times (A^* \times A^*)$$

$$divide : A^* \times A^* \rightarrow \mathsf{B}\,(A^*, A, (A^* \times A^*))$$

$$\mathsf{B}\,(Z, X, Y) = Z + X \times Y$$

```
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp> :{
*Cp|
*Cp| divide (l,[])
*Cp| divide ([],r)
*Cp| divide (x:xs,y:ys) | x
*Cp|                    | otherwise = i2 (y,(x:xs,ys))
*Cp|
*Cp| :}
*Cp>
*Cp> :t divide
divide :: Ord a => ([a], [a]) -> Either [a] (a, ([a], [a]))
*Cp>
```

work — ghc -B/Library/Frameworks/GHC.framework/Versions/8.2.2-x86_64/usr/lib/ghc-8.2.2 --interactive Cp — 80×21

~/work — vi a.hs | ...ramework/Versions/8.2.2-x86_64/usr/lib/ghc-8.2.2 --interactive Cp

```
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp>
*Cp> :{
*Cp|
*Cp| divide (l,[])
*Cp| divide ([],r)
*Cp| divide (x:xs,y:ys) | x
*Cp|                    | otherwise = i2 (y,(x:xs,ys))
*Cp|
*Cp| :}
*Cp>
*Cp> :t divide
divide :: Ord a => ([a], [a]
*Cp>
```

$$divide : A^* \times A^* \rightarrow A^* + A \times (A^* \times A^*)$$

$$divide : A^* \times A^* \rightarrow \mathsf{B}\,(A^*, A, (A^* \times A^*))$$

$$\mathsf{B}\,(Z, X, Y) = Z + X \times Y$$

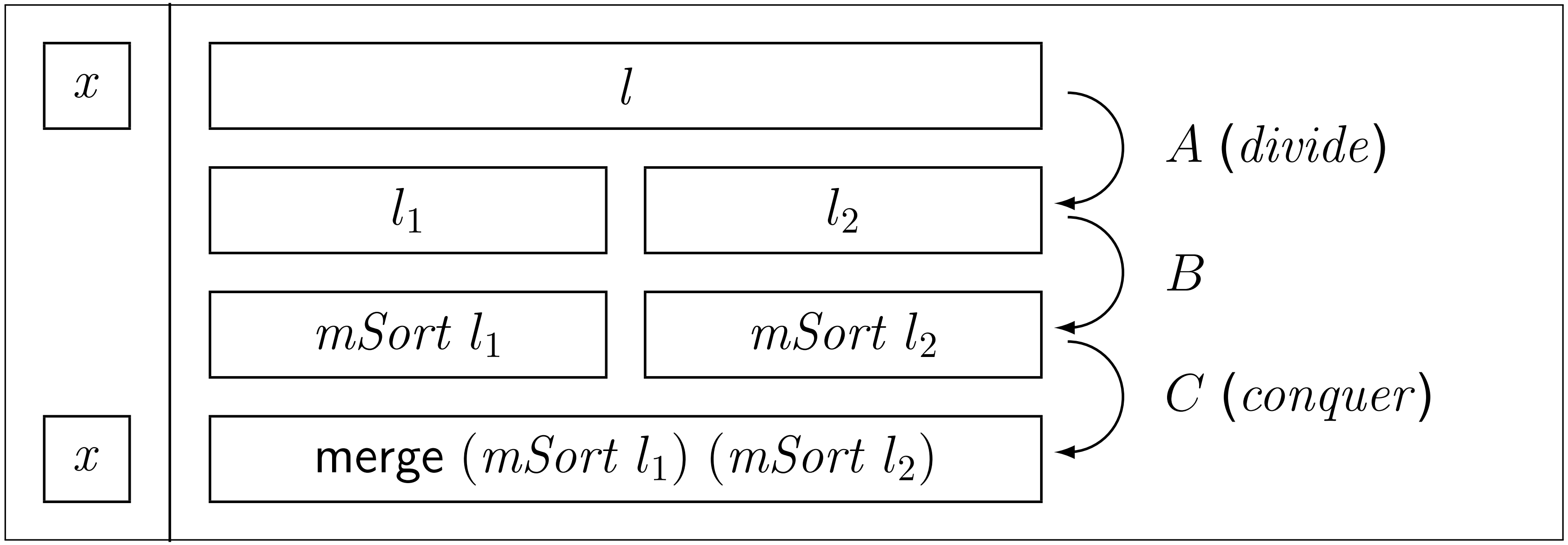
$$\mathsf{T}\,(Z, X) \cong \mathsf{B}\,(Z, X, \mathsf{T}\,(Z, X))$$

$$\mathsf{T}\,(Z, X) \cong Z + X \times \mathsf{T}\,(Z, X)$$

MERGESORT

a.hs

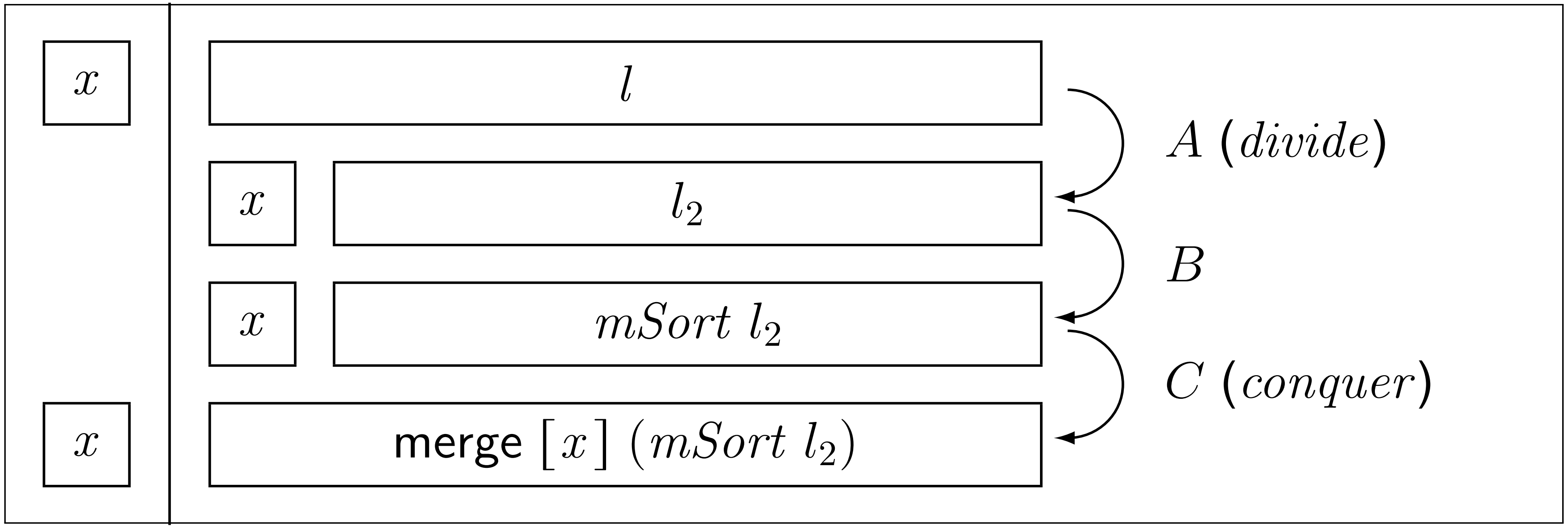
```haskell
mSort :: Ord a => [a] -> [a]
mSort [] = []
mSort [x] = [x]
mSort l = let (l1,l2) = sep l
          in merge(mSort l1, mSort l2)
```



MERGESORT

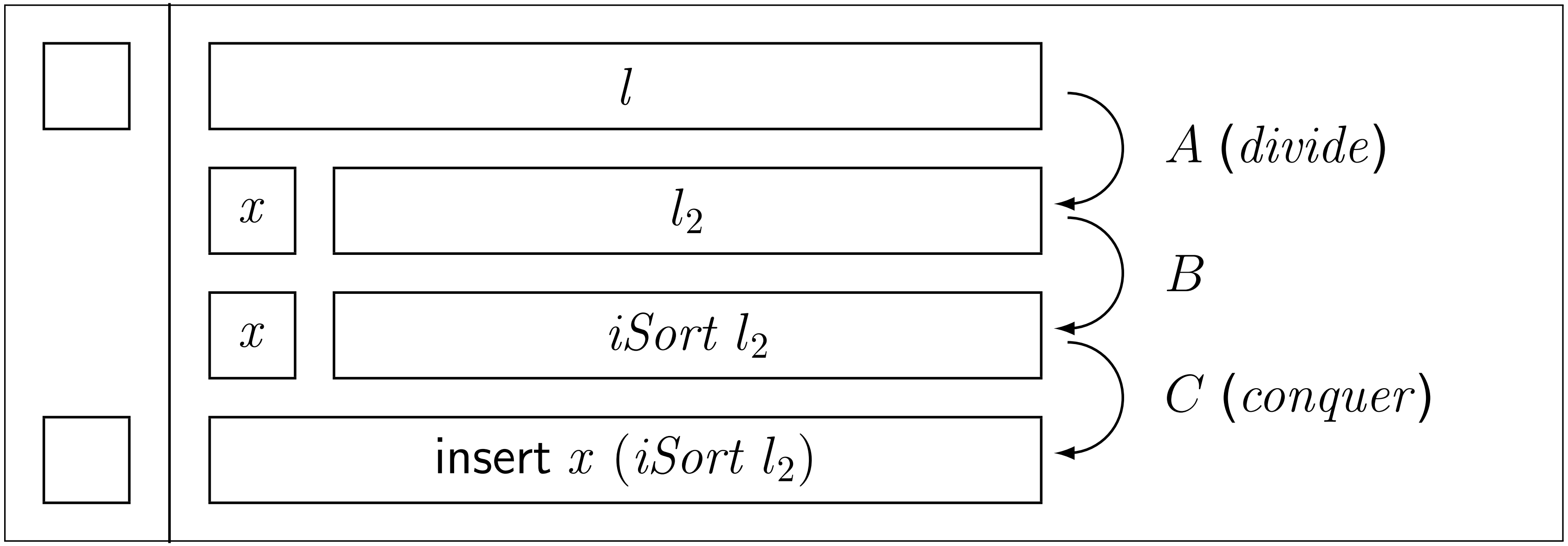
```haskell
merge :: Ord a => ([a], [a]) -> [a]
merge ([x],[])                    = [x]
merge ([],r)                      = r
merge ([x],y:ys) | x < y          = x : merge([],y:ys)
                 ) | otherwise = y : merge(x:[],ys
```

Caso particular:

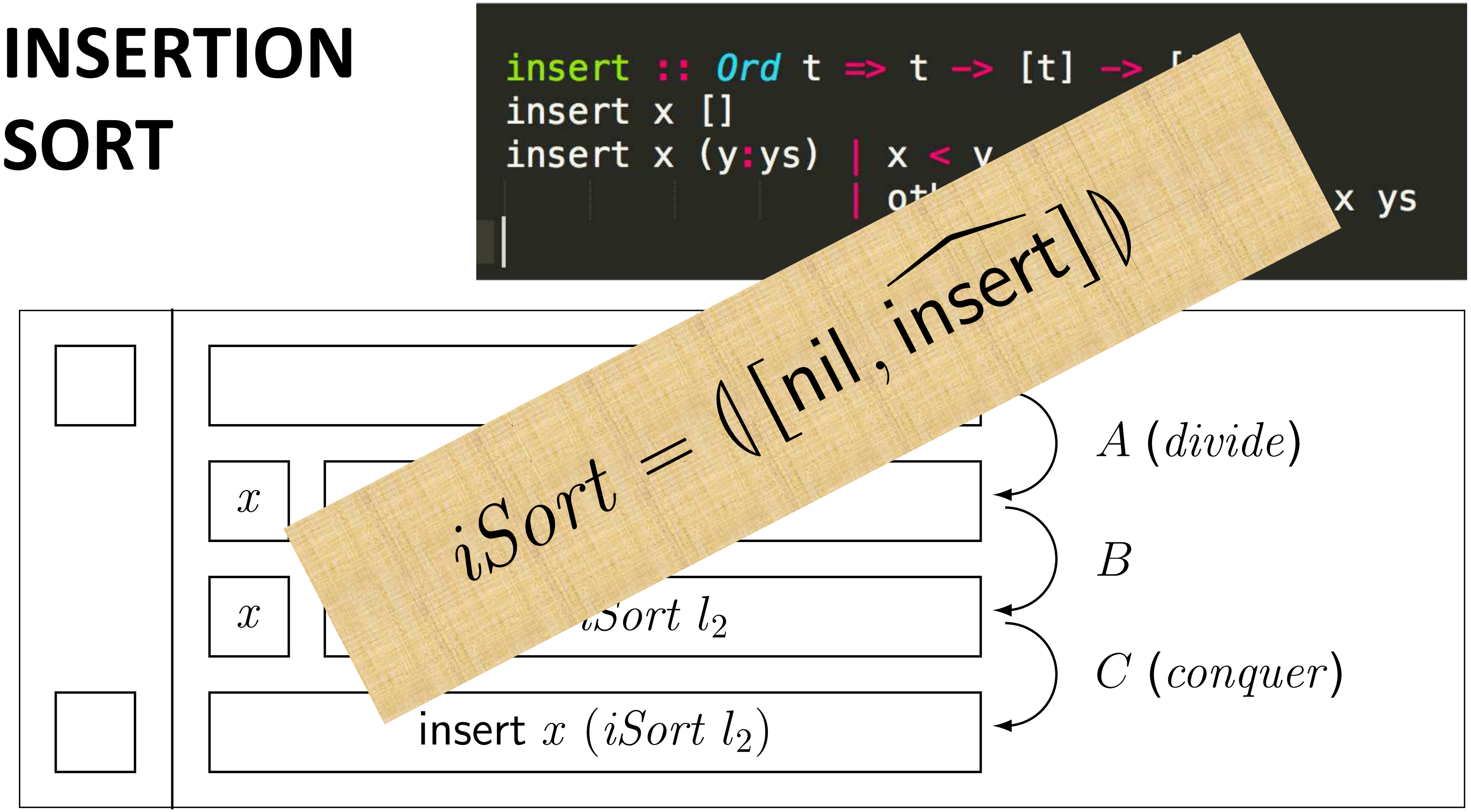


INSERTION SORT

```haskell
insert :: Ord t => t -> [t] -> [t]
insert x []                        = [x]
insert x (y:ys) | x < y            = x:y:ys
                | otherwise        = y:insert x ys
```



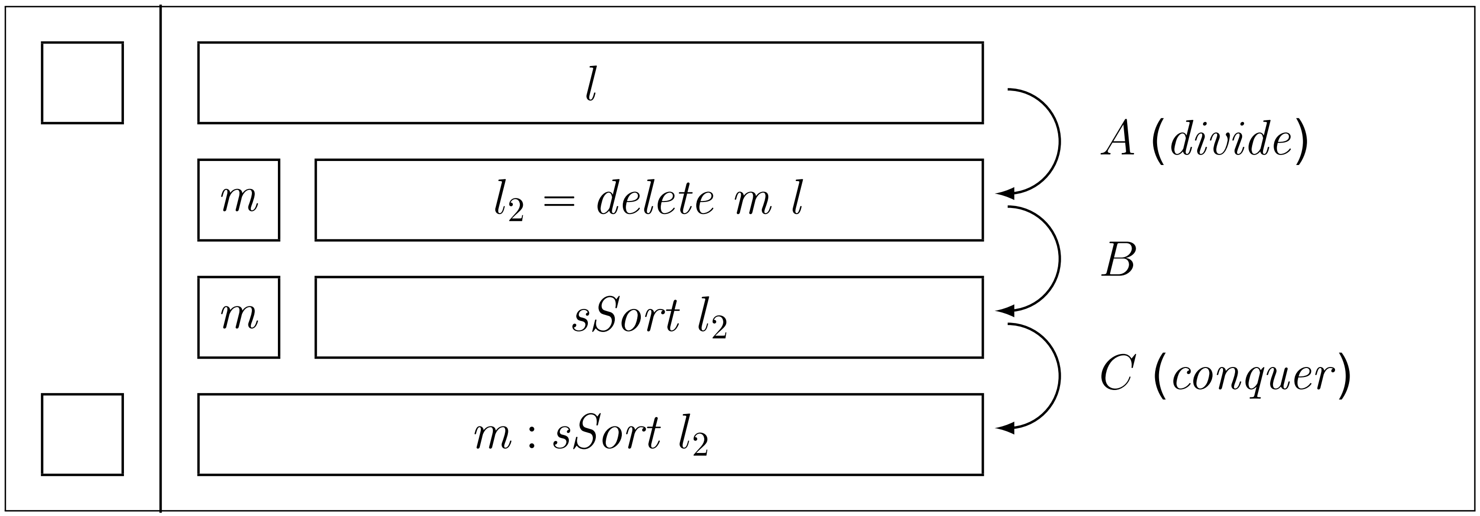
INSERTION SORT



SELECTION SORT

```haskell

sSort :: Ord a => [a] -> [a]
sSort = anaL divide where
   divide [] = i1()
   divide (xs) = let m = minimum xs
                 in  i2(m, delete m xs)
```

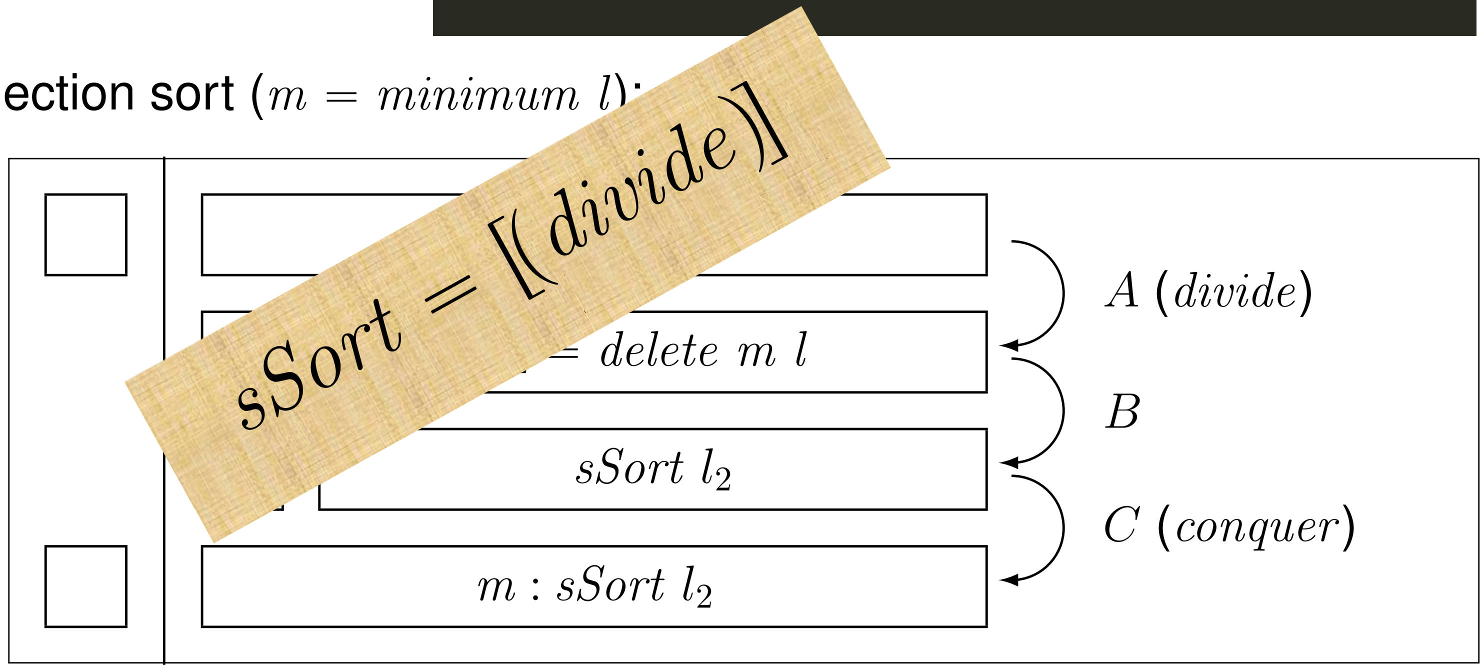
Selection sort ($m = minimum\ l$):



SELECTION SORT

```haskell

sSort :: Ord a => [a] -> [a]
sSort = anaL divide where
   divide [] = i1()
   divide (xs) = let m = minimum xs
                 in  i2(m, delete m xs)
```

Selection sort ($m = minimum\ l$):



ANA, CATA & HILO

$$C \xleftarrow{(\!| f |\!)} \mathsf{T} \xleftarrow{[\!(g)\!]} A$$

$$[\![f , g]\!] = (\!| f |\!) \cdot [\!(g)\!]$$

ANA, CATA & HILO

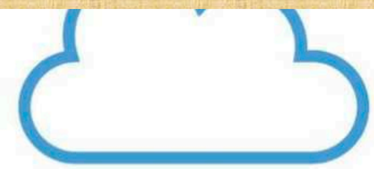
Leis de reflexão:

$$(\!|\, \mathsf{in} \,|\!) = id$$

$$[\!(\, \mathsf{out} \,)\!] = id$$

$$[\![\mathsf{in}, g]\!] = [\!(g)\!]$$

$$[\![f, \mathsf{out}]\!] = (\!|\, f \,|\!)$$



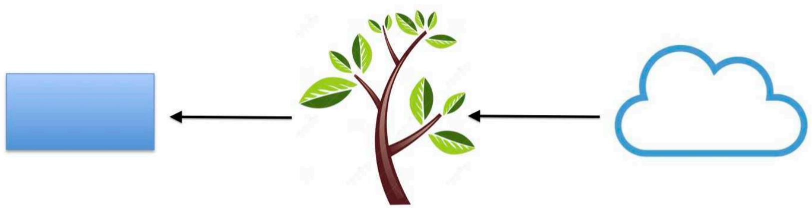
$$C \xleftarrow{(\!|\, f \,|\!)} \mathsf{T} \xleftarrow{[\!(g)\!]} A$$

$$[\![f, g]\!] = (\!|\, f \,|\!) \cdot [\!(g)\!]$$

CLASSIFICAÇÃO

	Singleton	Equal-size
Easy Split/Hard Join	Insertion Sort	Merge Sort
Hard Split/Easy Join	Selection Sort	Quick Sort

NB:
'Split' = *divide*
'Join' = *conquer*



$$C \xleftarrow{(\!|\_|\!)} \mathsf{T} \xleftarrow{[\!(\_)\!]} A$$

Research at Google

Lecture: **The Google MapReduce**

http://research.google.com/archive/mapreduce.html

10/03/2014

Romain Jacotin

romain.jacotin@orange.fr

Research at Google

Lecture: The Google MapReduce

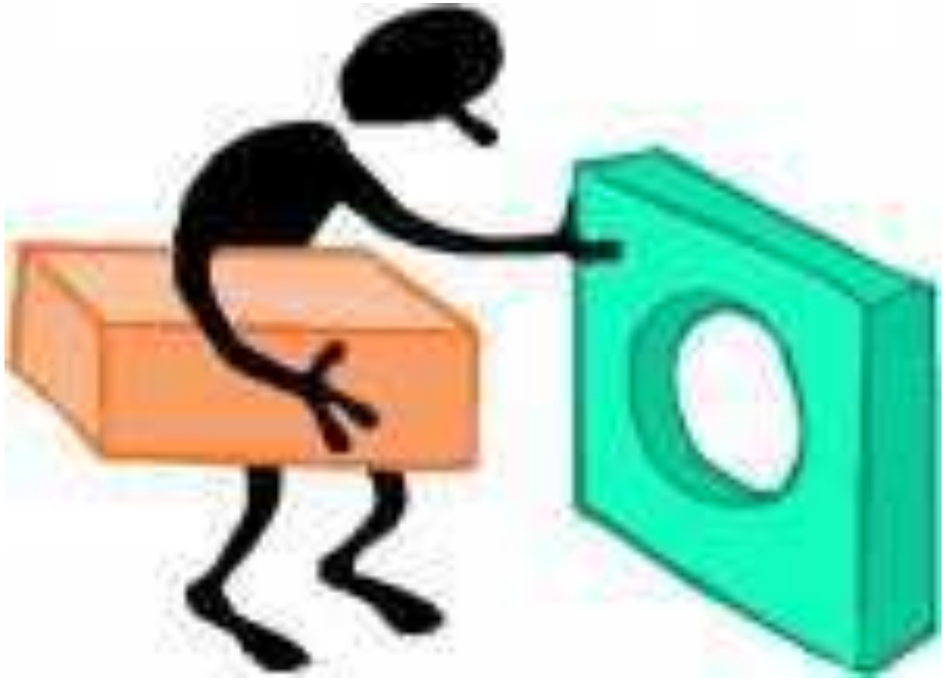
$$(\!| g |\!) \cdot \mathbf{T} f = (\!| g \cdot \mathbf{B}(f, id) |\!)$$

http://research.google.com/archive/mapreduce.html

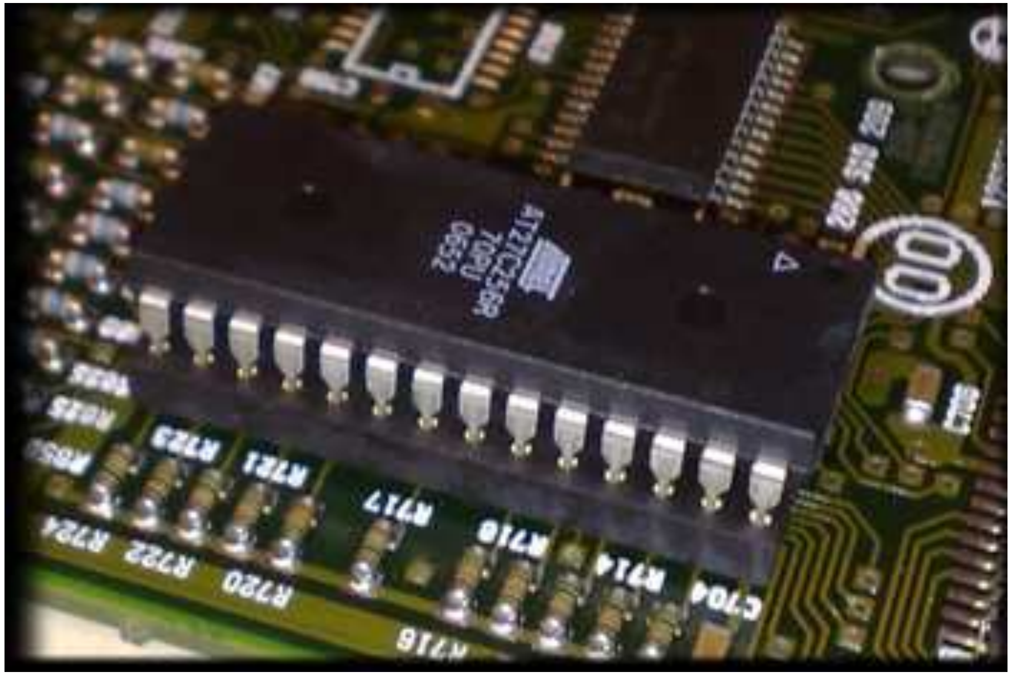
10/03/2014

Romain Jacotin
romain.jacotin@orange.fr

(PARAMETRIC) TYPE CHECKING



Componentes de “Hardware” / “Software”



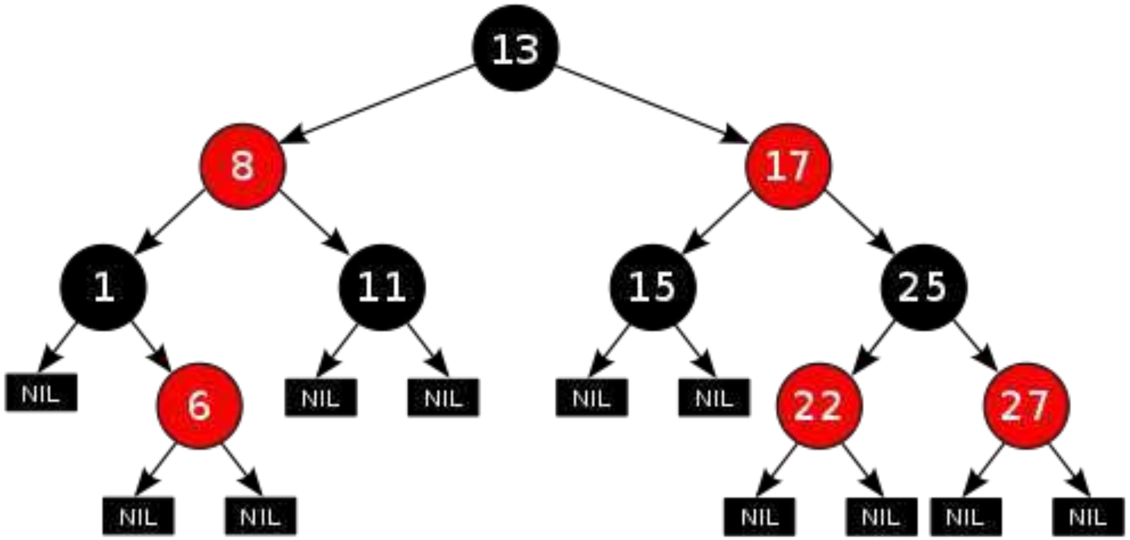
Software
Reuse

Software Reuse

Cálculo de Programas
Aula T11

Pesquisa binária

```haskell
lookBTree :: Ord a => a -> BTree (a,b) -> Maybe b
lookBTree a Empty = Nothing
lookBTree a (Node((a',b'),(l,r)))
        | a == a' = Just b'
        | a < a' =  lookBTree a l
        | a > a' =  lookBTree a r
```



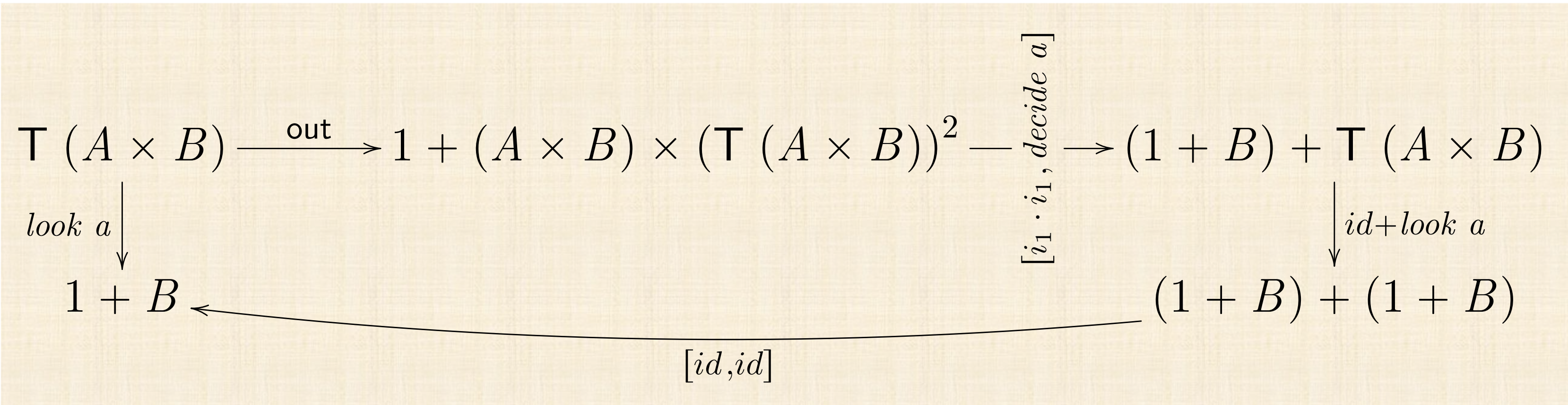
Pesquisa binária

```haskell
lookBTree :: Ord a => a -> BTree (a,b) -> Maybe b
lookBTree a Empty = Nothing
lookBTree a (Node((a',b'),(l,r)))
    | a == a' = Just b'
    | a < a' =  lookBTree a l
    | a > a' =  lookBTree a r
```

		B (X, Y)	$1 + Y$	$X + Y$	$1 + X \times Y$	$Z + X \times Y$	$X + Y^2$	$1 + X \times Y^2$
A	C	T X	$\mathbb{N}_0$	XNat X	X^*	SList X Z	LTree X	BTree X
$\mathbb{N}_0$	$\mathbb{N}_0$	Factorial			fac		$dfac$	
$\mathbb{N}_0$	$\mathbb{N}_0$	Misc. em $\mathbb{N}_0$	$(n*), (n+), \_^n$ etc		sq		dsq, fib	
$\mathbb{N}_0$	$\mathbb{N}_0{}^*$	Séries			$odds$, $evens$			
$\mathbb{N}_0 \times X^*$	X^*	Selecção		$udrop$	$utake$			
$\mathbb{R}$	$\mathbb{R}$	Raiz quadrada		$\sqrt{-}_\epsilon$				
X^*	X^*	Filtragem			filter p	filter p		
X^*	X^*	Ordenação	$bSort$		$iSort$, $sSort$		$mSort$	$qSort$
X^*	X^{**}	Grupos			$chunksOf$ n			
$X^* \times X^*$	X^*	Junção				merge, $uconc$		
$X \times X^*$	X^*	Inserção				insert		
$\mathbb{B} \times \mathbb{N}_0$	$(\mathbb{N}_0 \times \mathbb{B})^*$	Puzzles						$hanoi$
BTree (X, Y)	$1 + Y$	Look-up		$lookup$ x				
T (X, Y)	$1 + Y$	Look-up			$lookup$ x			
T X	T X	Inversão			$reverse$		mirror	mirror
T X	$\mathbb{N}_0$	Cardinalidades			length		$count$	$count$
T X	$\mathbb{N}_0$	Profundidades					$depth$	$depth$
T X	X^*	Travessias					$tips$	$inordt$, $preordt$, $posordt$
T X	X^{**}	Caminhos			$prefixes$, $sufixes$			$traces$
T (T X)	T X	'Multiplicação'			μ		μ	

Pesquisa binária

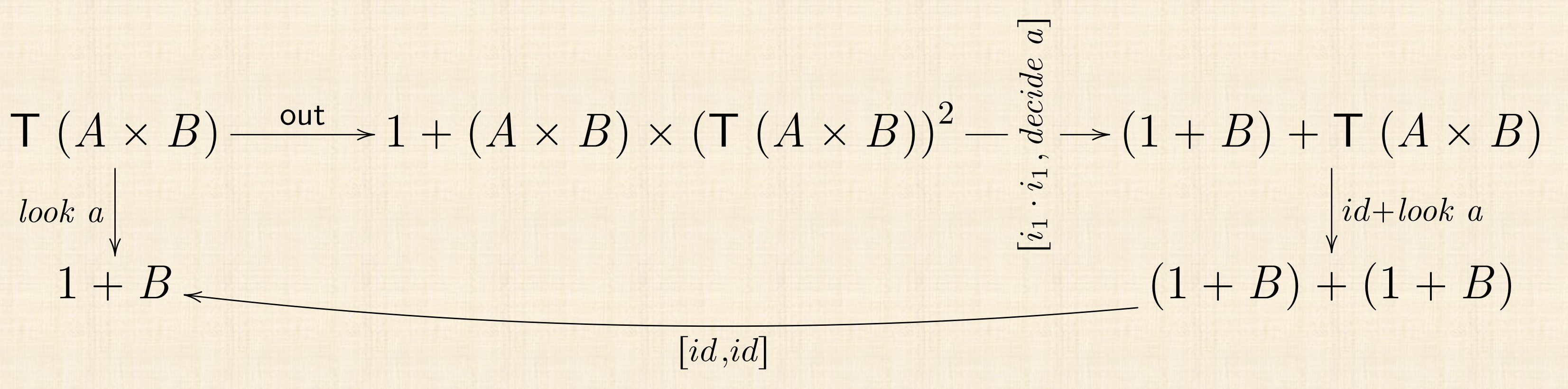
```haskell
lookBTree :: Ord a => a -> BTree (a,b) -> Maybe b
lookBTree a Empty = Nothing
lookBTree a (Node((a',b'),(l,r)))
      | a == a' = Just b'
      | a < a' =  lookBTree a l
      | a > a' =  lookBTree a r
```



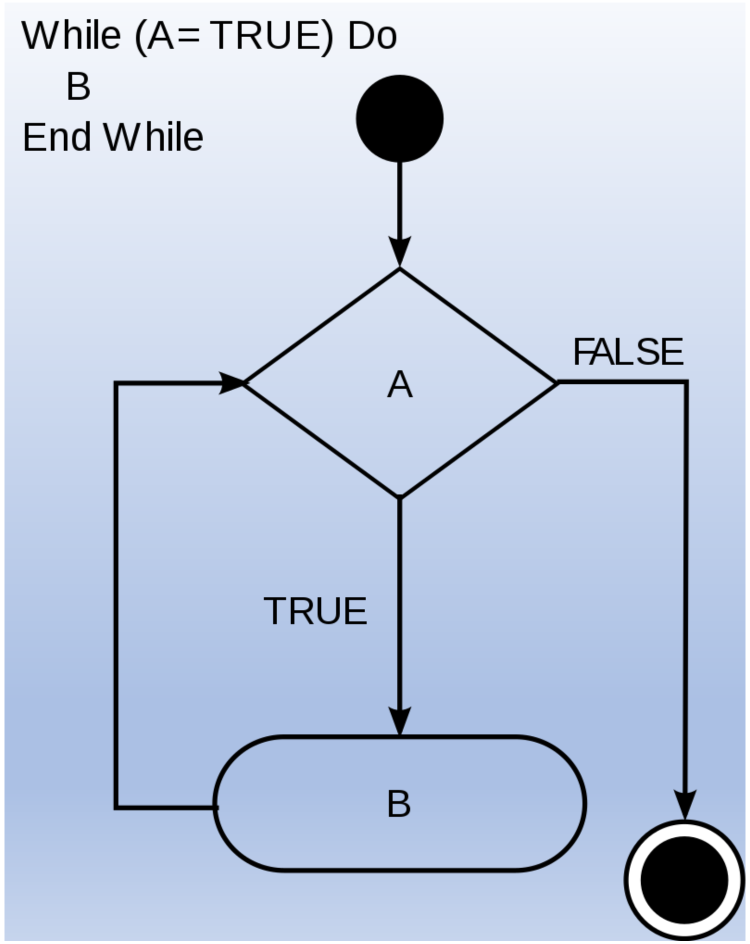
Pesquisa binária

$$
\begin{array}{l}
decide\ a\ ((a', b'), (l, r)) \\
\quad \mid a = a' = i_1\ (i_2\ b') \\
\quad \mid a < a' = i_2\ l \\
\quad \mid a > a' = i_2\ r
\end{array}
$$

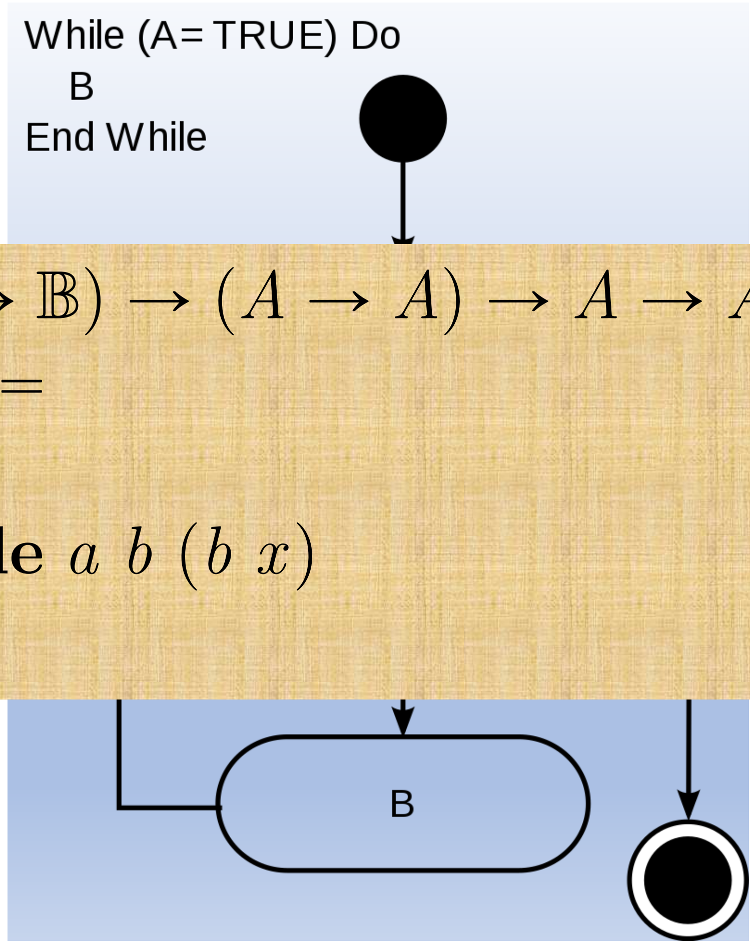
```haskell
lookBTree :: Ord a => a -> BTree (a,b) -> Maybe b
lookBTree a Empty = Nothing
lookBTree a (Node((a',b'),(l,r)))
      | a == a' = Just b'
      | a < a' =  lookBTree a l
      | a > a' =  lookBTree a r
```



‘While loop’



‘While loop’

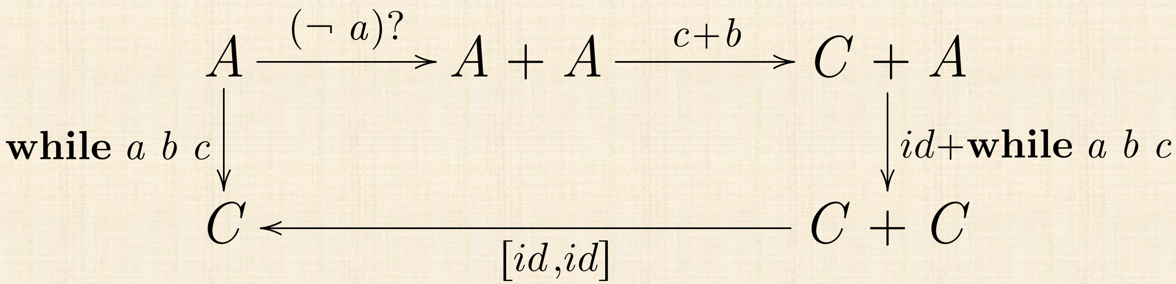


$$\mathbf{while} : (A \to \mathbb{B}) \to (A \to A) \to A \to A$$

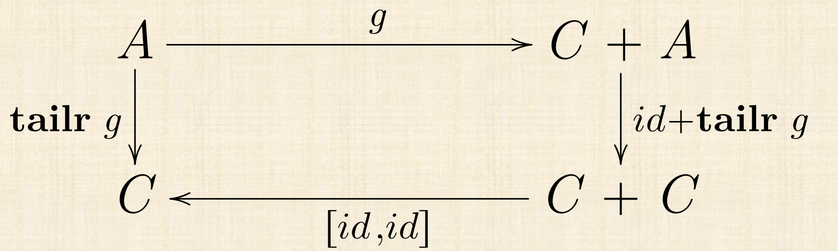
$$\begin{aligned}
&\mathbf{while}\ a\ b\ x = \\
&\quad \mathbf{if}\ a\ x \\
&\quad \mathbf{then\ while}\ a\ b\ (b\ x) \\
&\quad \mathbf{else}\ x
\end{aligned}$$

‘While loop’

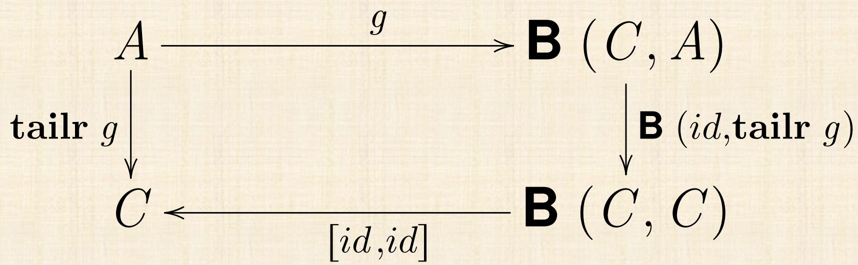
$$\begin{aligned}
&\mathbf{while} :: (A \to \mathbb{B}) \to (A \to A) \to (A \to C) \to A \to C \\
&\mathbf{while}\ a\ b\ c\ x = \\
&\quad \mathbf{if}\ a\ x \\
&\quad \mathbf{then\ while}\ a\ b\ c\ (b\ x) \\
&\quad \mathbf{else}\ c\ x
\end{aligned}$$



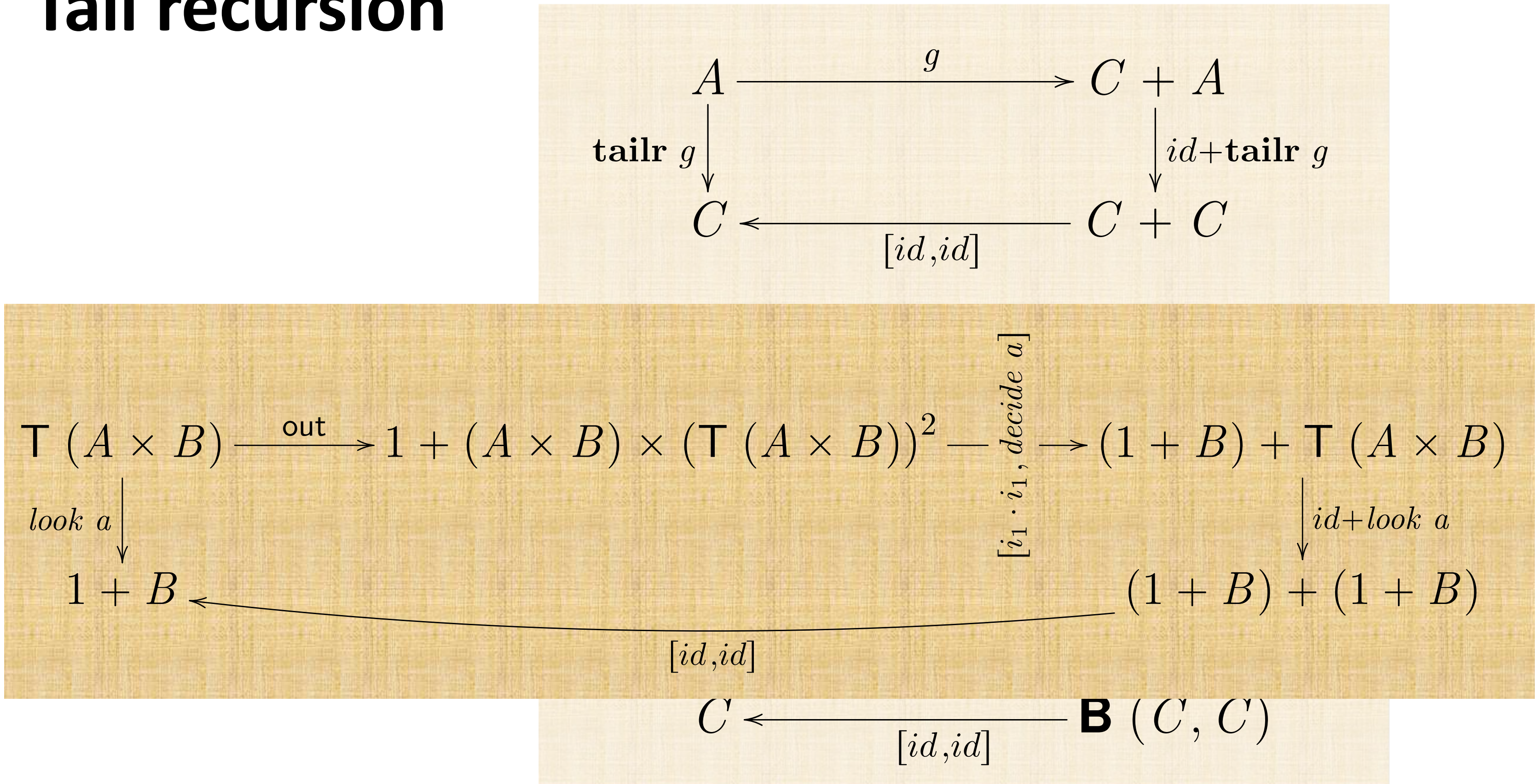
‘Tail recursion’



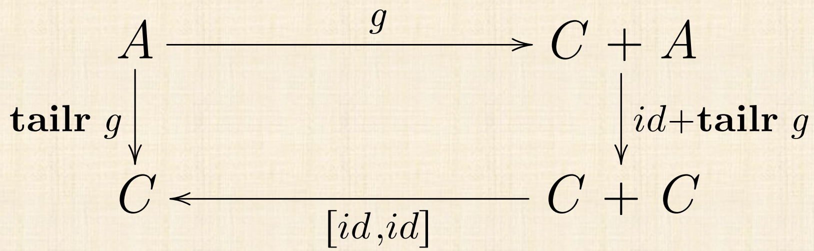
$$\mathbf{B}\,(X, Y) = X + Y$$



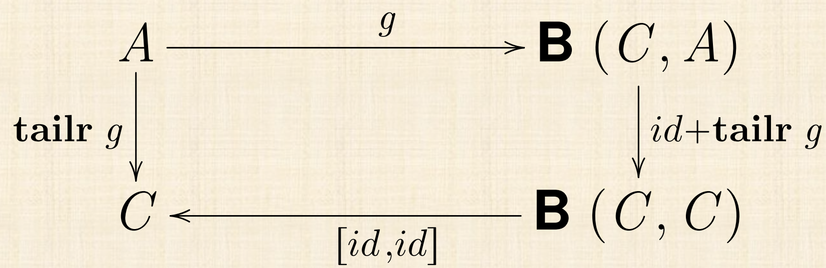
‘Tail recursion’



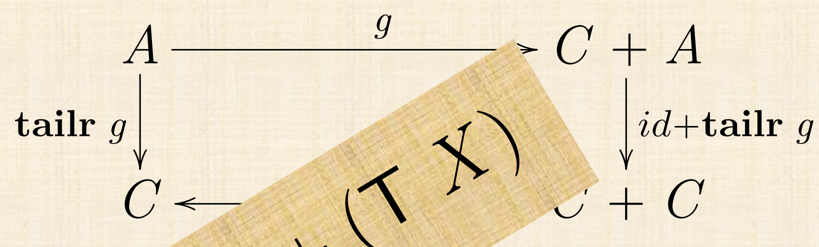
‘Tail recursion’



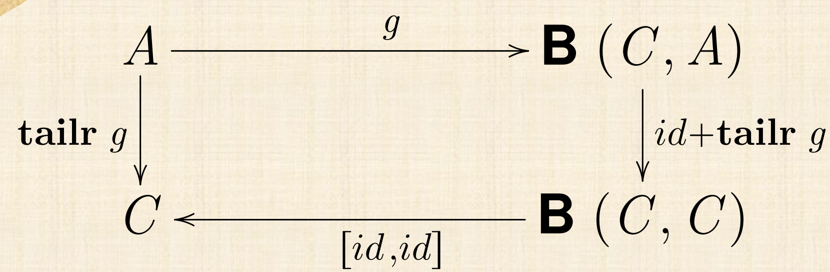
$$\mathbf{B}\,(X, Y) = X + Y$$



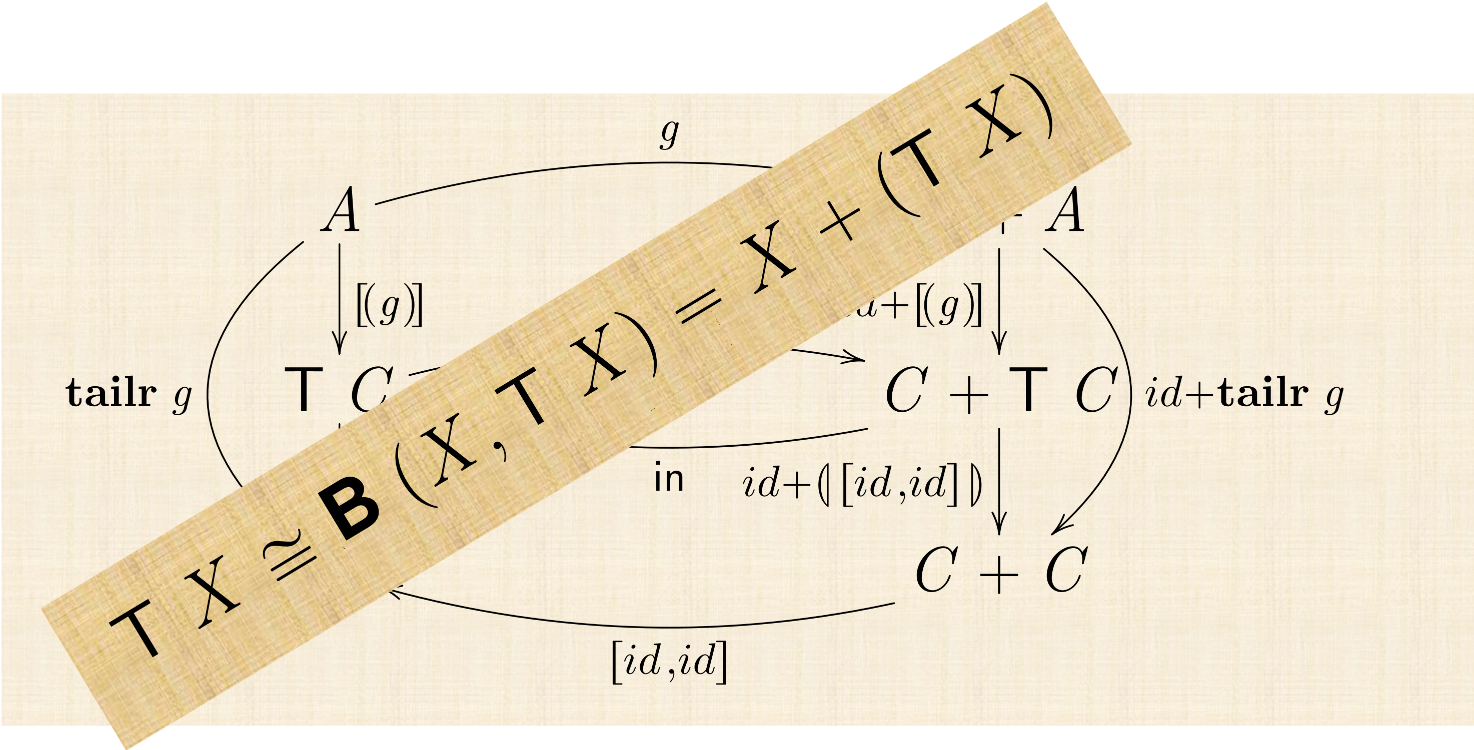
‘Tail recursion’



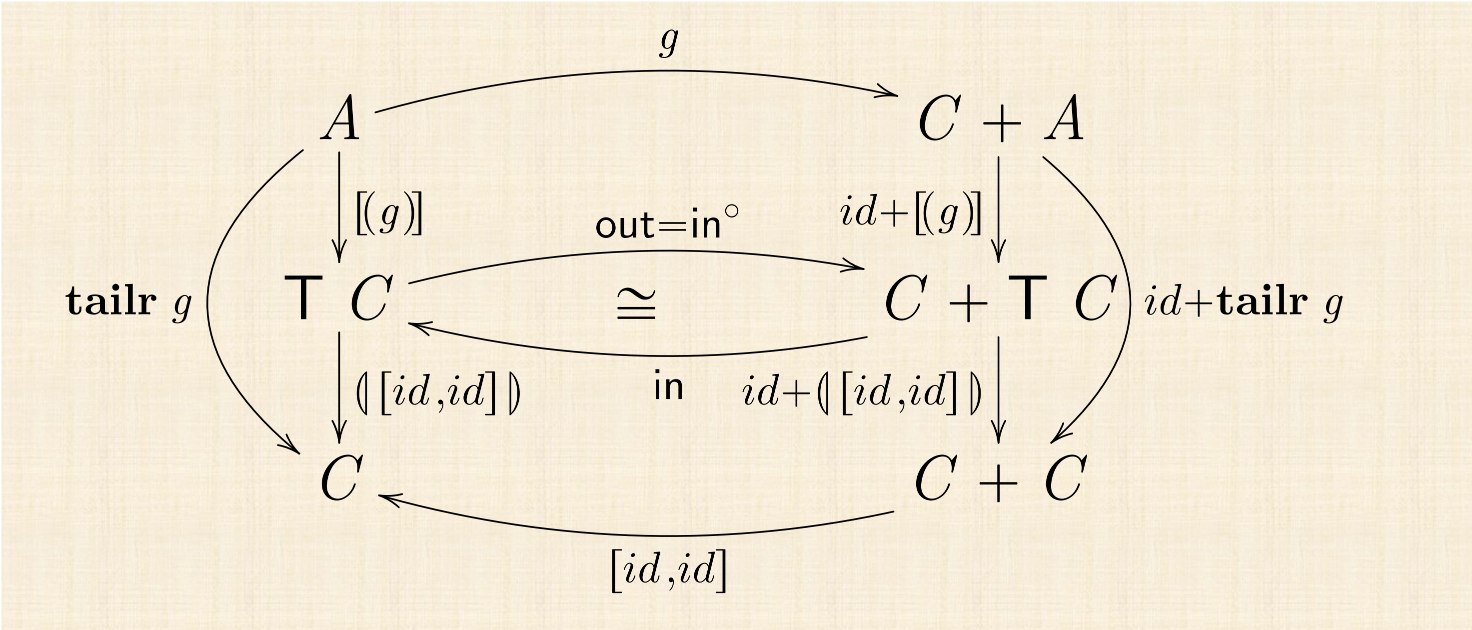
$$\mathsf{T}\,X \cong \mathbf{B}\,(X, \mathsf{T}\,X) = X + (\mathsf{T}\,X)$$



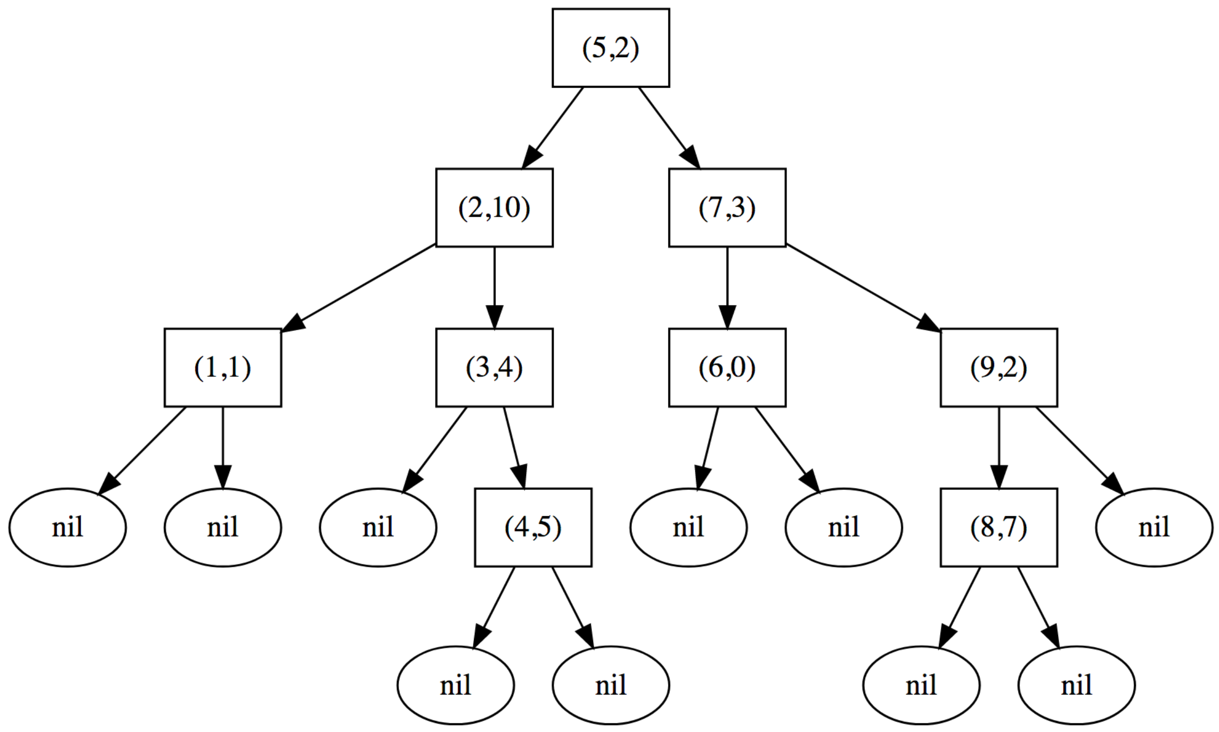
'Tail recursion' hylo



‘Tail recursion’



‘Tail recursion’



Funções pré-definidas

Função codiagonal: $codiag = [id, id]$

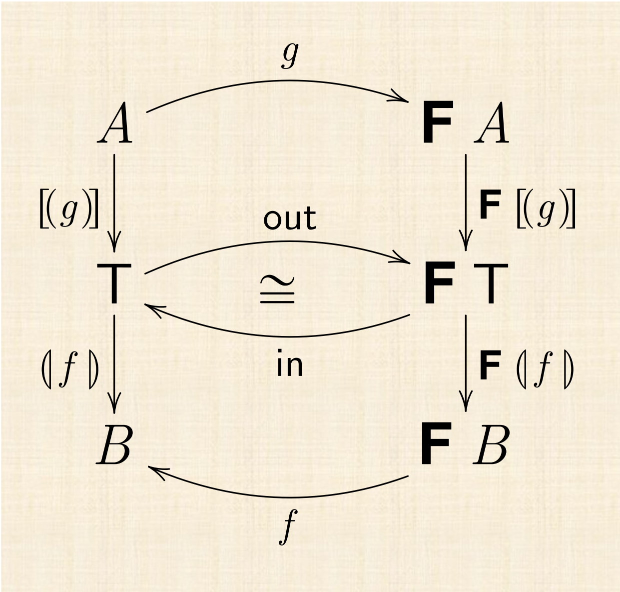
Função diagonal: $diag = \langle id, id \rangle$

‘Tail recursion’

```
let counter = 5;
let factorial = 1;

while (counter > 1)
    factorial *= counter--;

console.log(factorial);
```

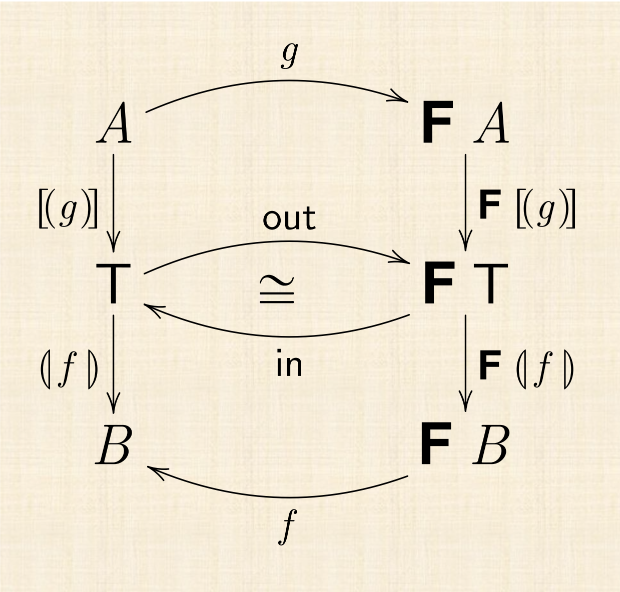
```
-- (4.4) Tail recursive factorial ---------------------------------------------

factr = while2 p f p2 . split id (const 1) where
   p(c,f) = c > 1
   f(c,f) = (c-1,c*f)
--------------------------------------------------------------------------------
```

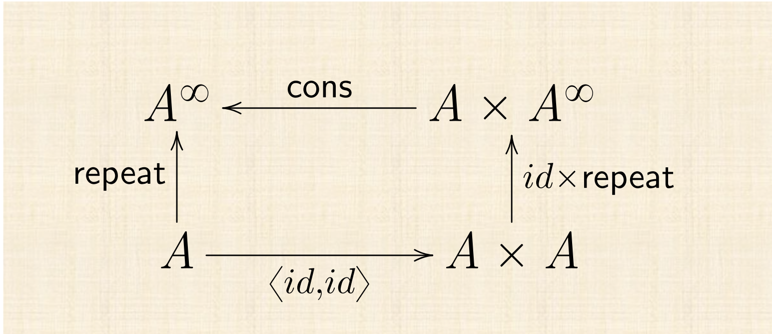
HILOMORFISMO



$$[\![f , g]\!] = (\!| f |\!) \cdot [\!(g)\!]$$

Algoritmos versus processos





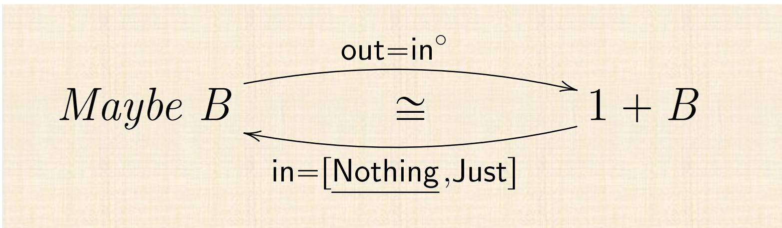
repeat $a = a$: repeat a

Função codiagonal: $codiag = [id, id]$

Função diagonal: $diag = \langle id, id \rangle$

Funções parciais

```haskell
lookBTree :: Ord a => a -> BTree (a,b) -> Maybe b
lookBTree a Empty = Nothing
lookBTree a (Node((a',b'),(l,r)))
        | a == a' = Just b'
        | a < a' =  lookBTree a l
        | a > a' =  lookBTree a r
```



PROGRAM DESIGN BY CALCULATION

4

WHY MONADS MATTER

In this chapter we present a powerful device in state-of-the-art functional programming, that of a *monad*. The monad concept is nowadays of primary importance in computing science because it makes it possible to describe computational effects as disparate as input/output, comprehension notation, state variable updating, probabilistic behaviour, context dependence, partial behaviour *etc.* in an elegant and uniform way.

Our motivation to this concept will start from a well-known problem in functional programming (and computing as a whole) — that of coping with undefined computations.

4

WHY MONADS MATTER

In this chapter we present a powerful device in tional programming, that of a *monad*. The monad of primary importance in computing science bec sible to describe computational effects as dispar comprehension notation, state variable updating, probabilistic behaviour, context dependence, partial behaviour *etc.* in an elegant and uniform way.

Our motivation to this concept will start from a well-known problem in functional programming (and computing as a whole) — that of coping with undefined computations.

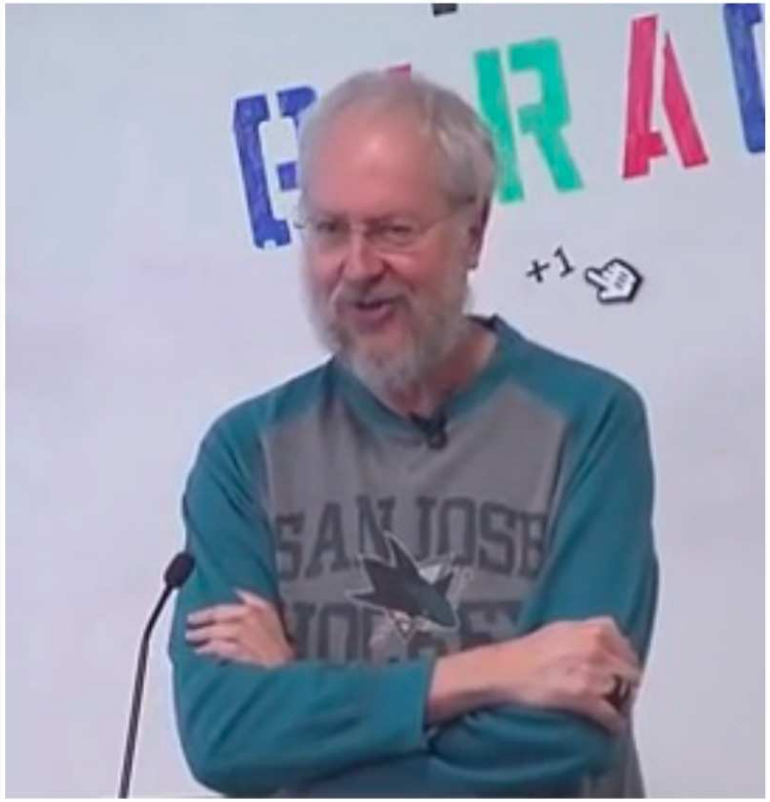

CHANCE

Probabilidade da soma

“Os *monads* [...] vêm com uma maldição. A maldição monádica é que,
quando uma pessoa aprende o que são mônadas e como usá-las,
perde a capacidade de explicar isso para outras pessoas!”

"Monads [...] come with a curse. The monadic curse is that once someone learns what monads are and how to use them, they lose the ability to explain it to other people"

(Douglas Crockford: *Google Tech Talk on how to express monads in JavaScript* YouTube 2013)



Douglas Crockford (2013)

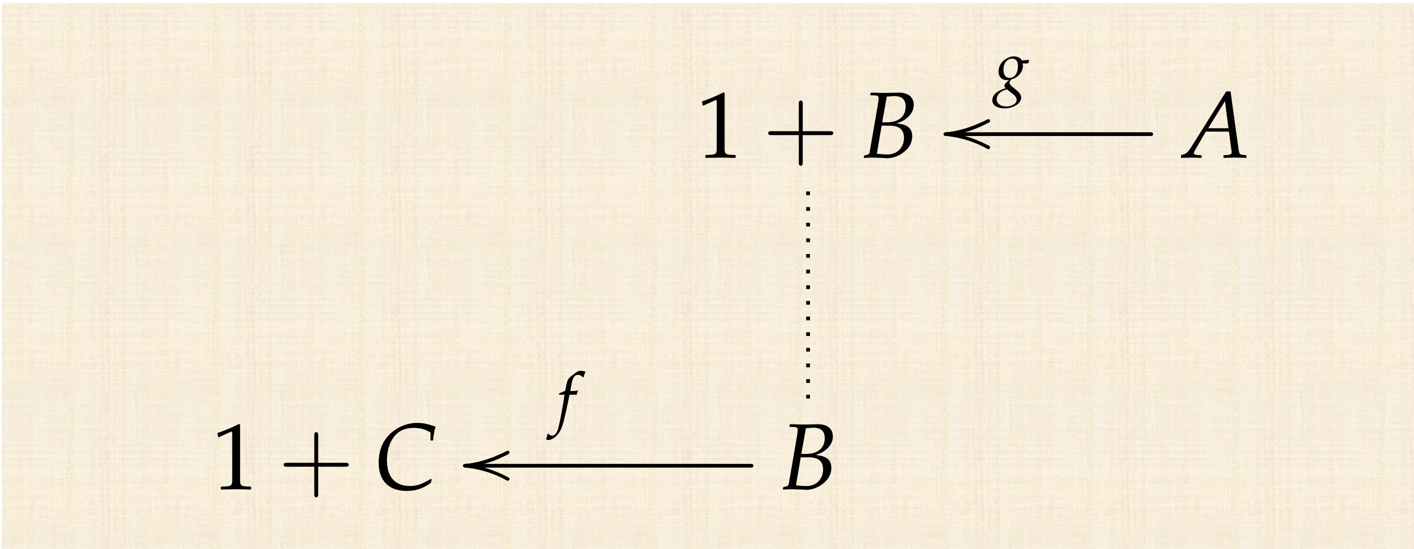
Funções parciais

$$1 + B \xleftarrow{\quad g \quad} A$$

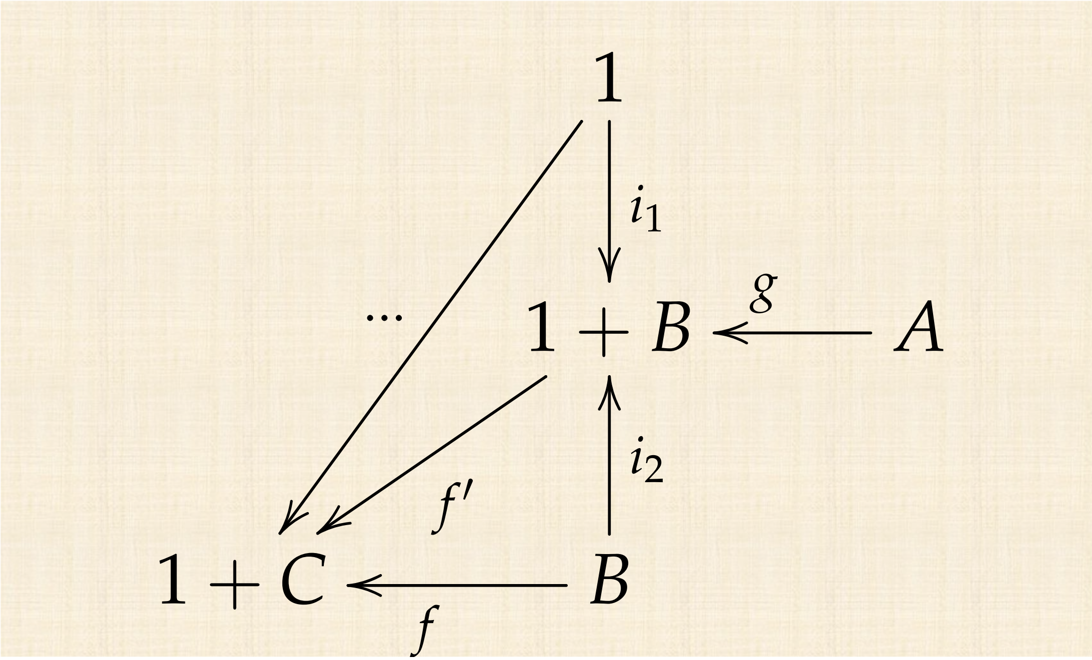
$$B \xleftarrow{\quad f \quad} A$$

$$1 + B \xleftarrow{\quad i_2 \cdot f \quad} A$$

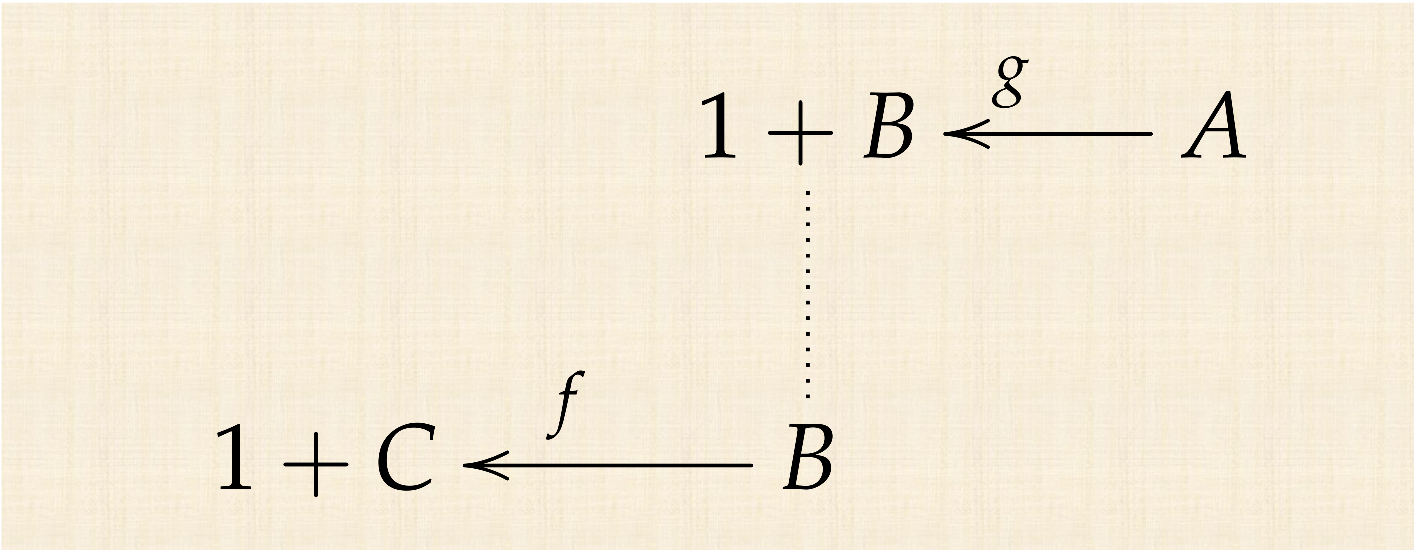
Funções parciais



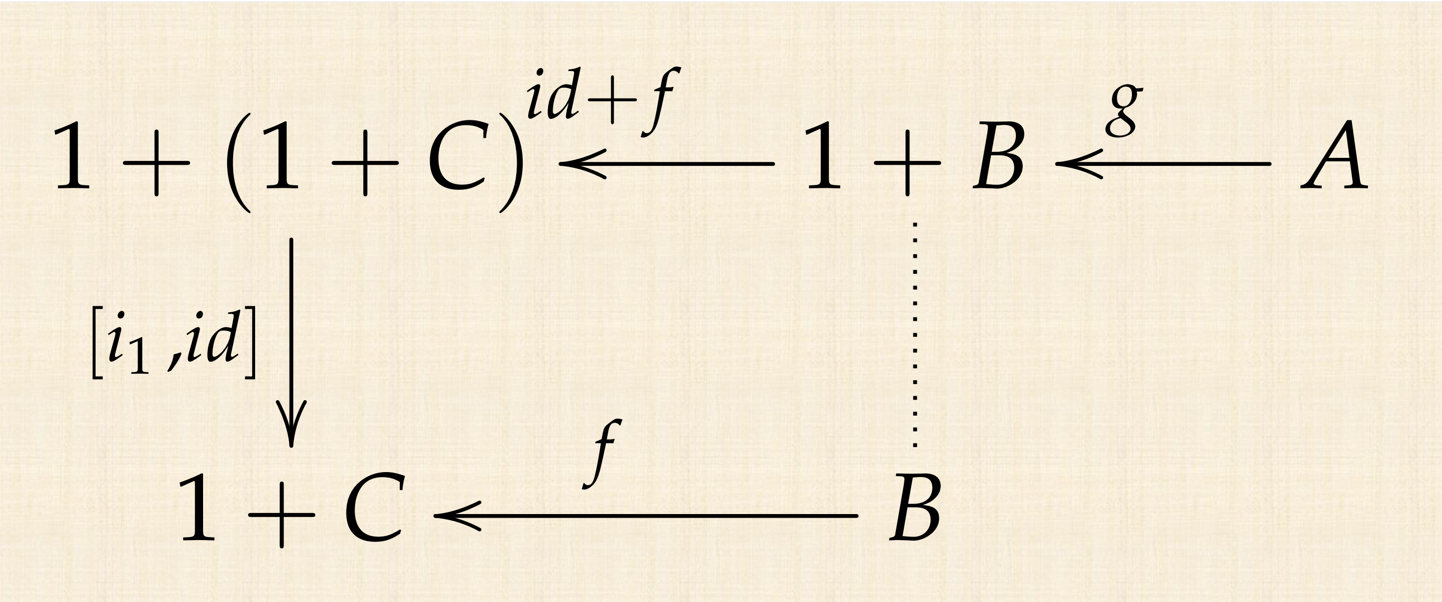
Funções parciais



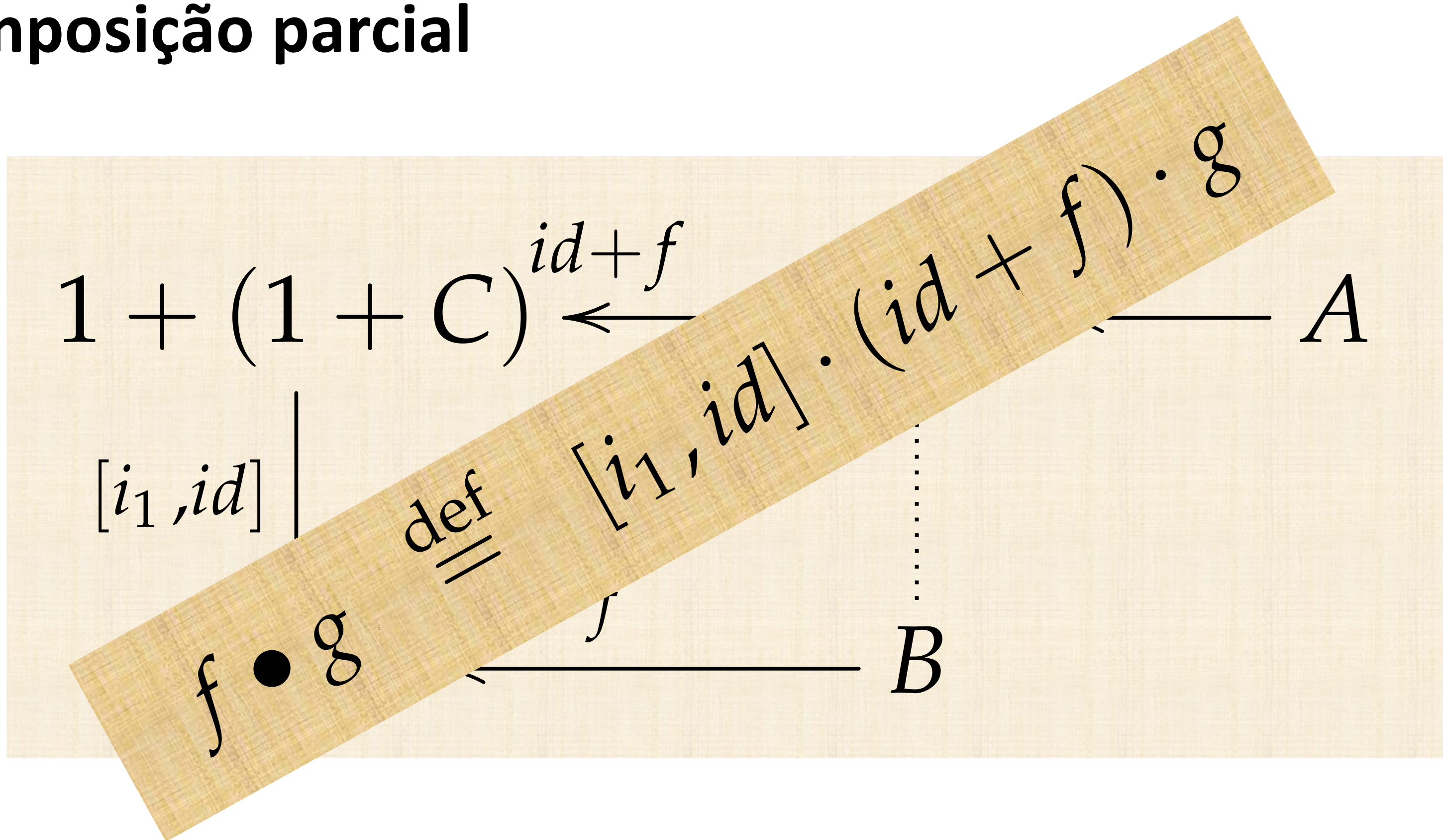
Composição parcial



Composição parcial

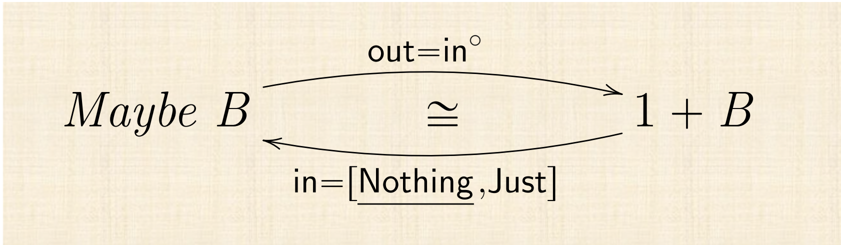


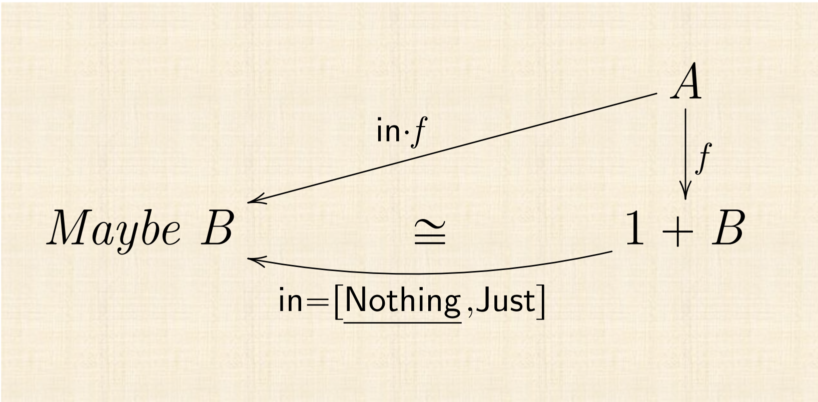
Composição parcial



$$f \bullet g \stackrel{\text{def}}{=} [i_1, id] \cdot (id + f) \cdot g$$

Funções ‘Maybe’

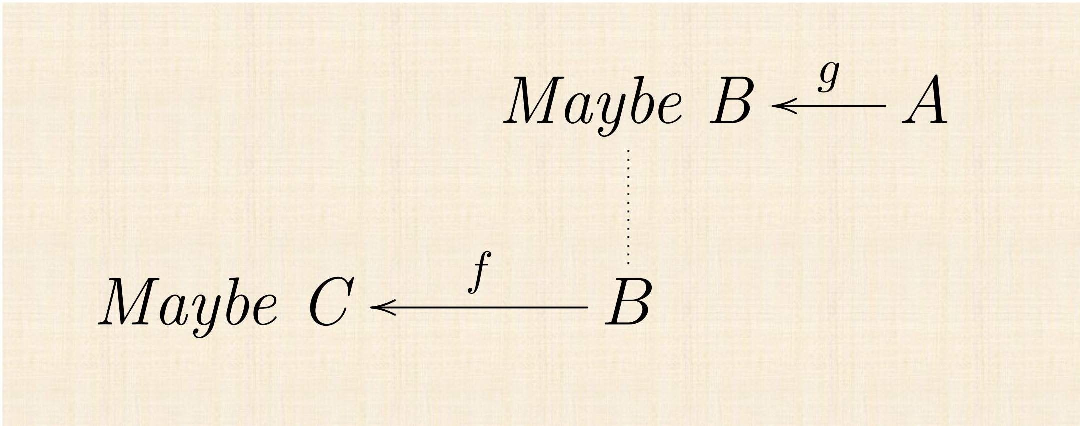




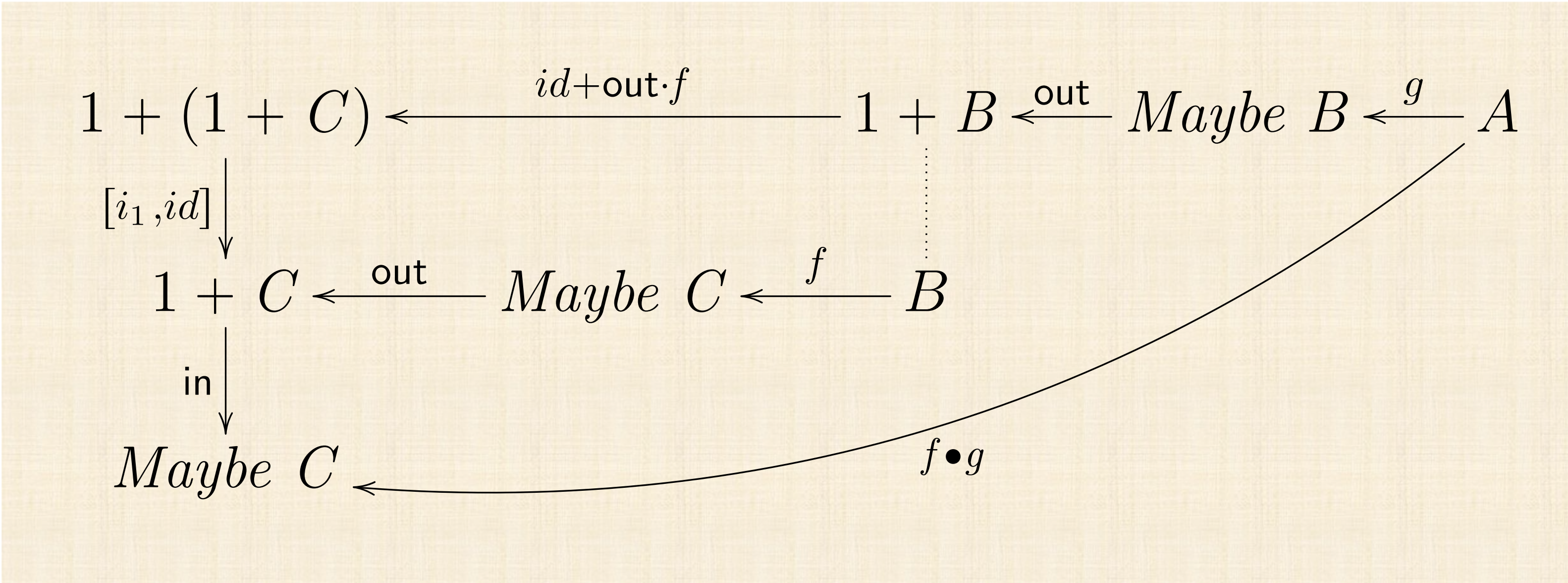
$$\begin{aligned} \mathsf{out}\ \mathsf{Nothing} &= i_1\ () \\ \mathsf{out}\ (\mathsf{Just}\ a) &= i_2\ a \end{aligned}$$

```
data Maybe a = Nothing | Just a
```

Composição de funções "Maybe"



Composição de funções “Maybe”



Composição de funções "Maybe"

$$f \bullet g = \mathsf{in} \cdot [i_1, \mathsf{out} \cdot f] \cdot \mathsf{out} \cdot g$$

$\equiv \quad \{$ fusão-+ e $\mathsf{in} \cdot \mathsf{out} = id$ $\}$

$$f \bullet g = [\mathsf{in} \cdot i_1, f] \cdot \mathsf{out} \cdot g$$

$\equiv \quad \{$ introdução da variável a $\}$

$$(f \bullet g)\ a = [\mathsf{in} \cdot i_1, f]\ (\mathsf{out}\ (g\ a))$$

$\equiv \quad \{$ definição de out $\}$

$$(f \bullet g)\ a = \mathbf{if}\ g\ a = \mathsf{Nothing}\ \mathbf{then}\ [\mathsf{in} \cdot i_1, f]\ (i_1\ ())\ \mathbf{else}\ [\mathsf{in} \cdot i_1, f]\ (i_2\ (g\ a))$$

$\equiv \quad \{$ cancelamento-+ e simplificação $\}$

$$(f \bullet g)\ a = \mathbf{if}\ g\ a = \mathsf{Nothing}\ \mathbf{then}\ \mathsf{Nothing}\ \mathbf{else}\ f\ (g\ a)$$

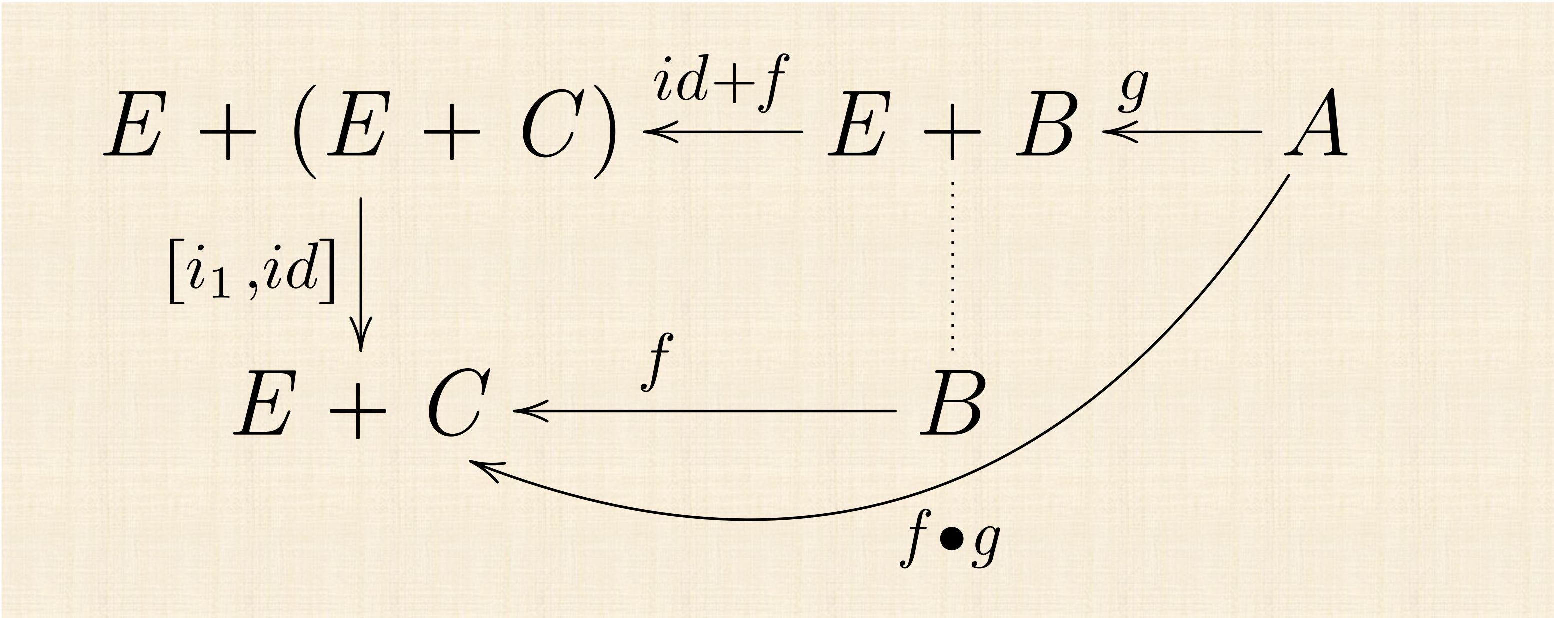
Funções com mensagens de erro

$$E + B \xleftarrow{\;g\;} A$$

$$B \xleftarrow{\;f\;} A$$

$$E + B \xleftarrow{\;i_2 \cdot f\;} A$$

Tratamento das mensagens de erro




Unchecked Exception
Checked Exception
Error
Exception Handling

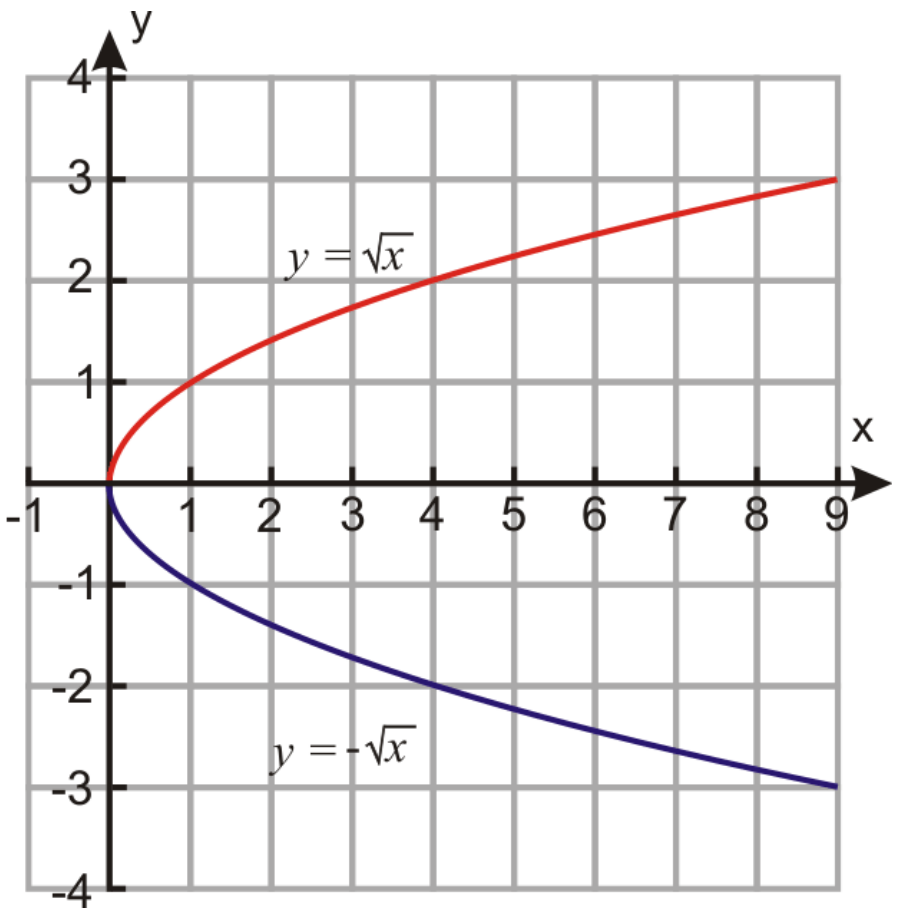
Funções ‘indecisas’

$$B^{\star} \xleftarrow{\;g\;} A$$

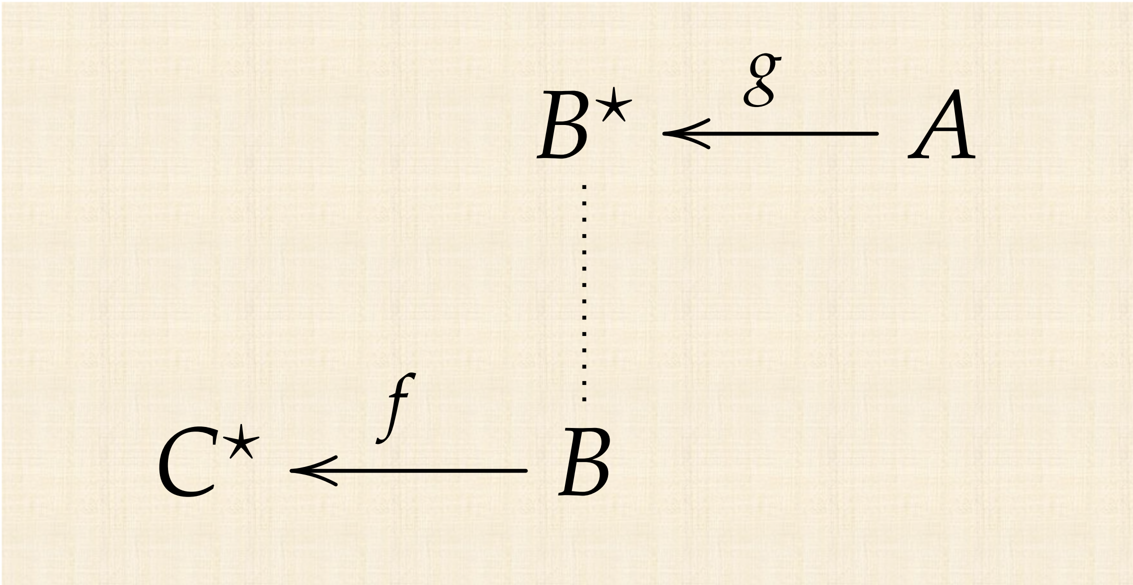
$$B \xleftarrow{\;f\;} A$$

$$B^{*} \xleftarrow{\;\mathsf{singl}\cdot f\;} A$$

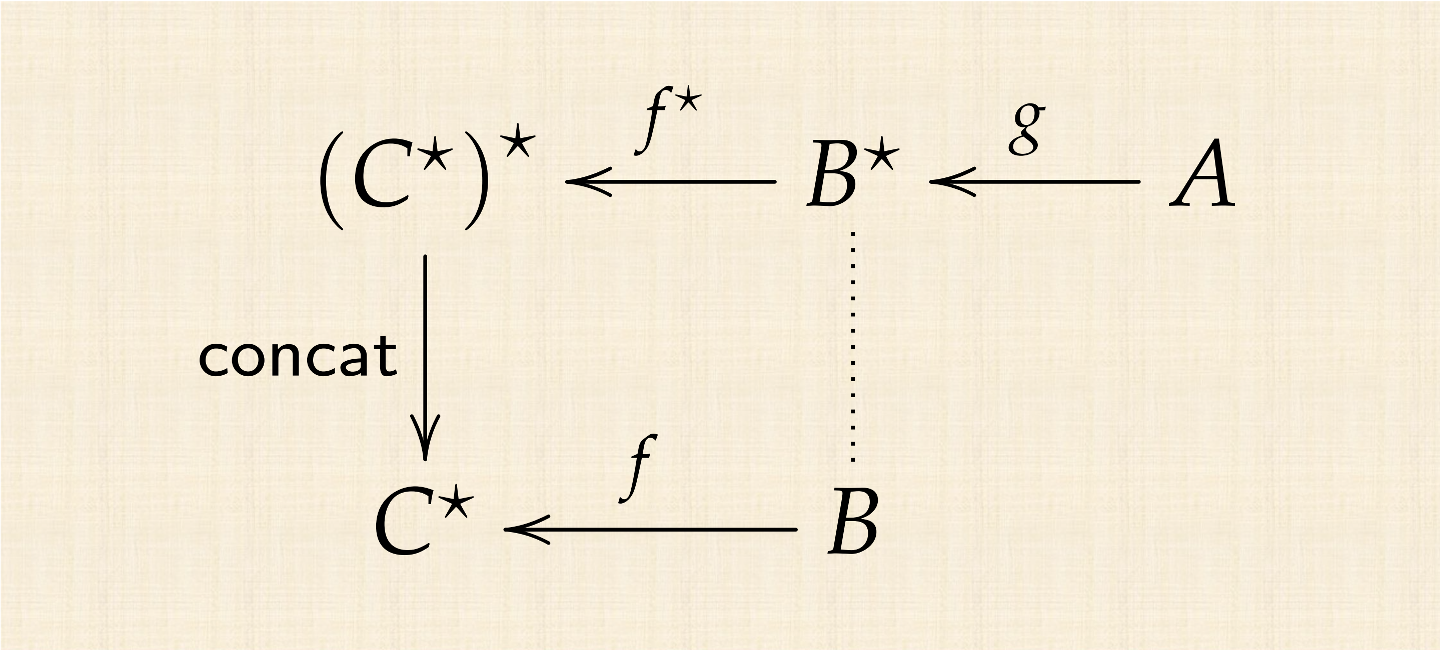
Raiz quadrada



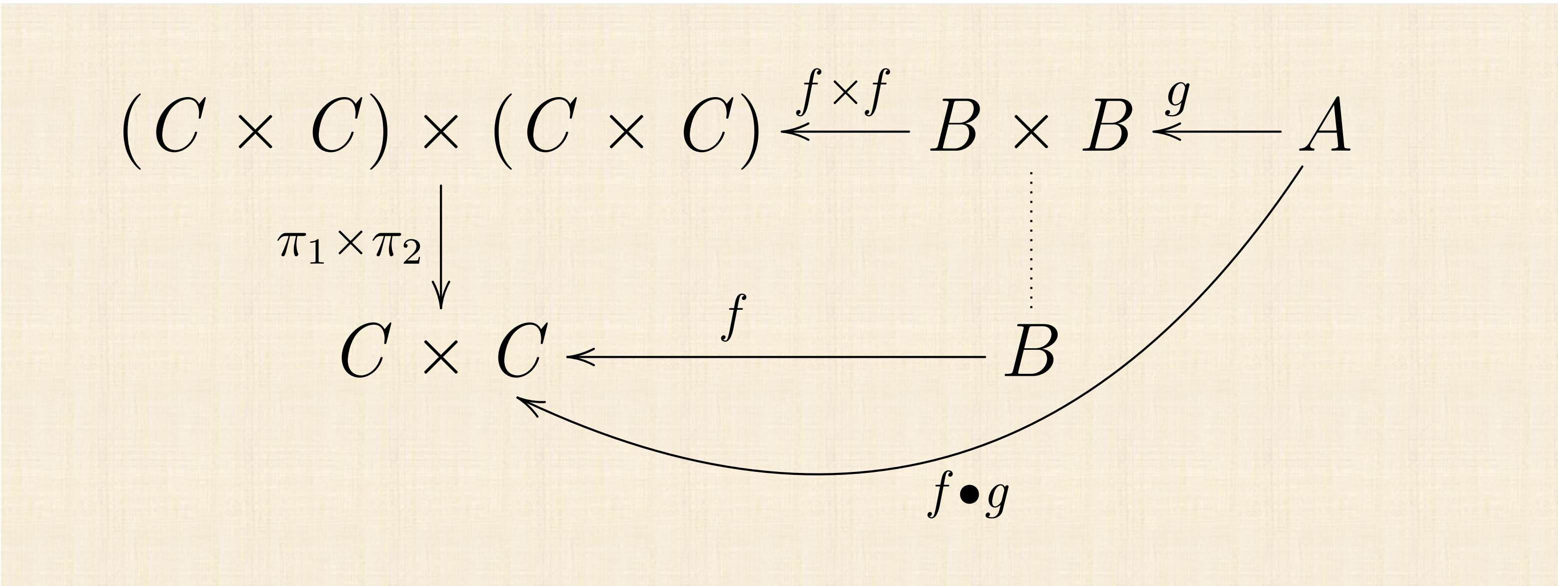
Composição de funções 'indecisas'



Composição de funções 'indecisas'



Composição de funções 'que dão pares'



FUNCTORES

$$\mathsf{T}\ X = 1 + X$$

$$\mathsf{T}\ X = Maybe\ X$$

$$\mathsf{T}\ X = E + X$$

$$\mathsf{T}\ X = X^*$$

$$\mathsf{T}\ X = X \times X$$

Estrutura semelhante

$$X \xrightarrow{i_2} 1 + X \xleftarrow{[i_1\,, id]} 1 + (1 + X)$$

$$X \xrightarrow{i_2} E + X \xleftarrow{[i_1\,, id]} E + (E + X)$$

$$X \xrightarrow{\mathsf{Just}} Maybe\ X \xleftarrow{\mu} Maybe\ (Maybe\ X)$$

$$X \xrightarrow{singl} X^* \xleftarrow{\mathsf{concat}} (X^*)^*$$

$$X \xrightarrow{\langle id, id \rangle} X \times X \xleftarrow{\pi_1 \times \pi_2} (X \times X) \times (X \times X)$$

MONAD

$$X \xrightarrow{u} \mathbf{T}\,X \xleftarrow{\mu} \mathbf{T}\,(\mathbf{T}\,X)$$

MONAD

$$X \xrightarrow{u} \mathbf{T}\, X \xleftarrow{\mu} \mathbf{T}\,(\mathbf{T}\, X)$$

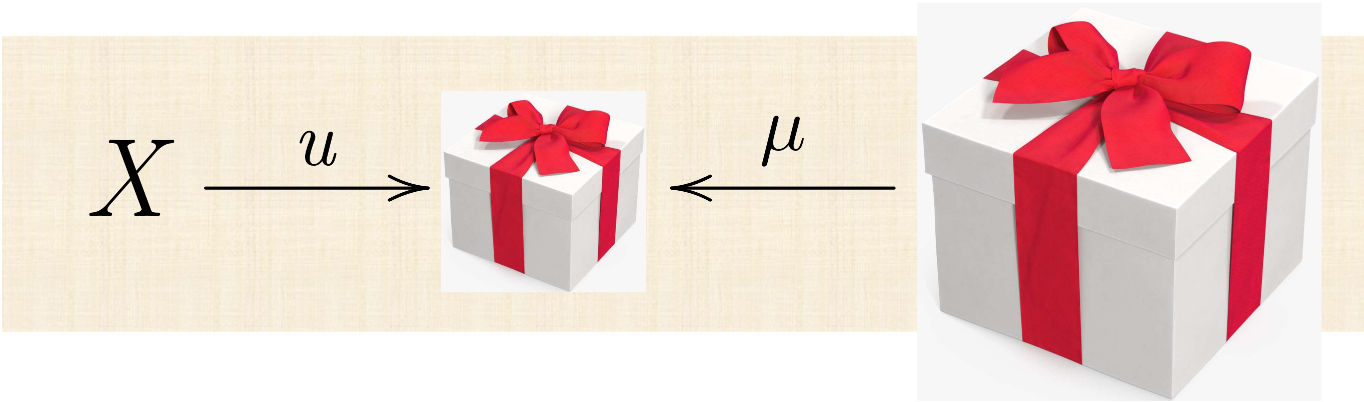
unidade multiplicação

MONAD

$$X \xrightarrow{\quad u \quad} \mathbf{T}\,X \xleftarrow{\quad \mu \quad} \mathbf{T}\,(\mathbf{T}\,X)$$

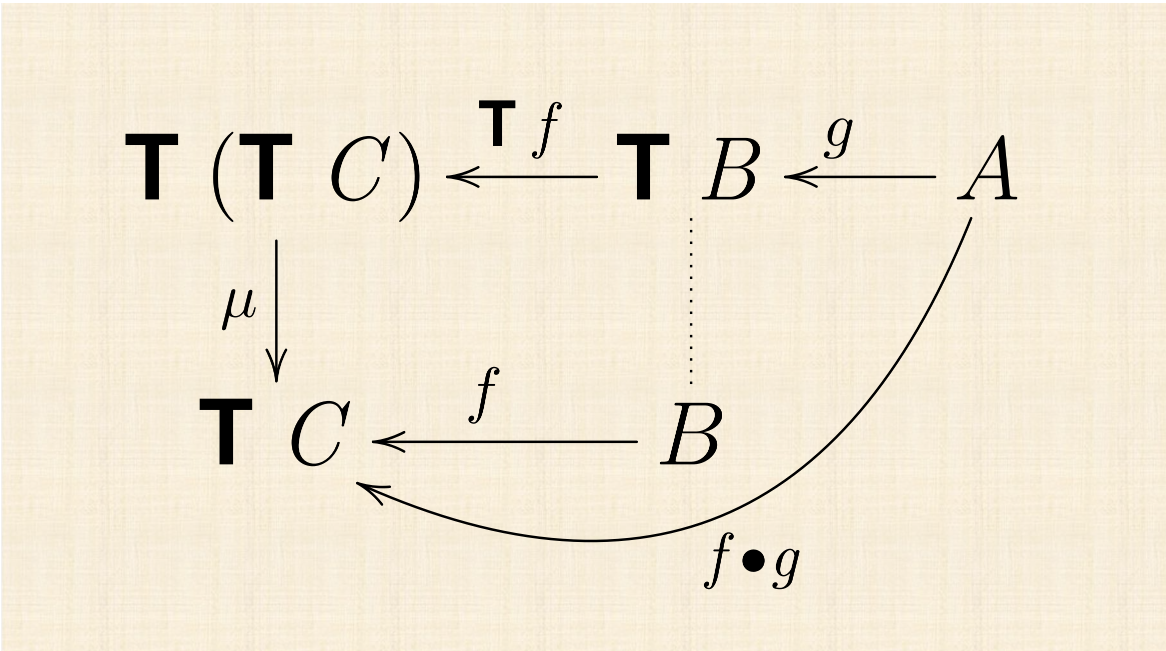
Monad = **Functor** + unidade + multiplicação

MONAD



Monad = **Functor** + unidade + multiplicação

Composição monádica



Heinrich Kleisli

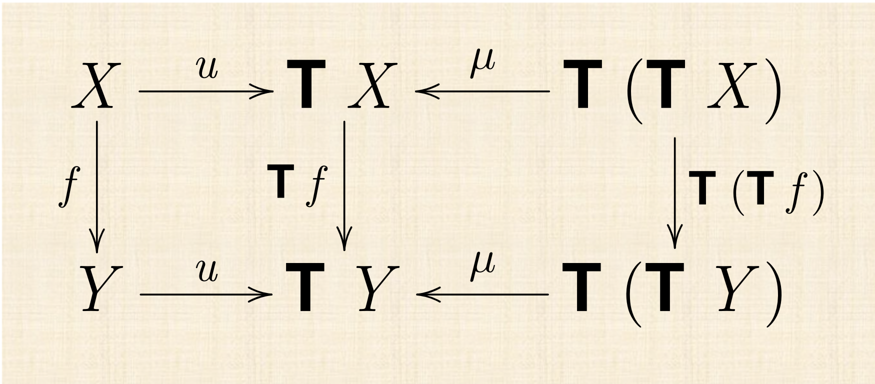
From Wikipedia, the free encyclopedia

Heinrich Kleisli (/ˈklaɪsli/; October 19, 1930 – April 5, 2011) was a Swiss mathematician. He is the namesake of several constructions in category theory, including the Kleisli category and Kleisli triples. He is also the namesake of the Kleisli Query System, a tool for integration of heterogeneous databases developed at the University of Pennsylvania.

Heinrich Kleisli in 1987

Kleisli earned his Ph.D. at ETH Zurich in 1960, having been supervised by Beno Eckmann and Ernst Specker. His dissertation was on homotopy and abelian categories. He served as an associate professor at the University of Ottawa before relocating to the University of Fribourg in 1966. He became a full professor at Fribourg in 1967.

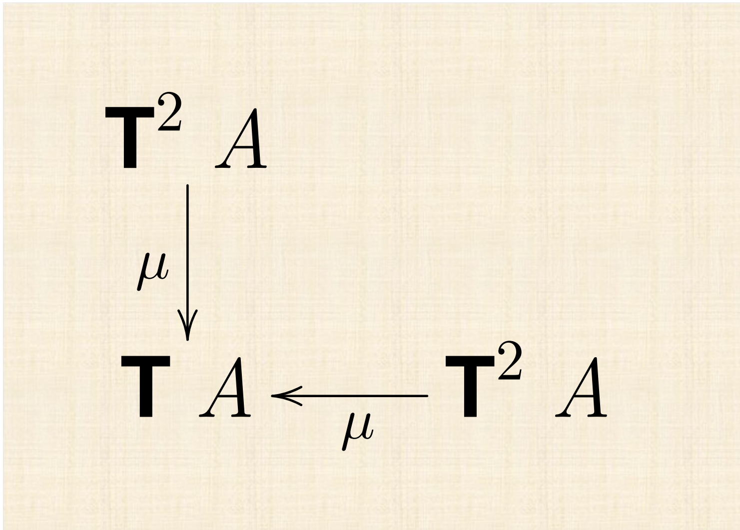
Monad – propriedades naturais



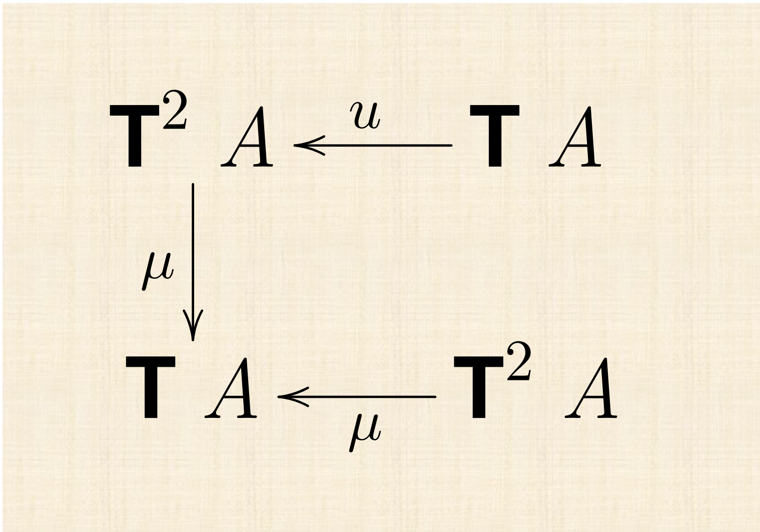
$$\mathsf{T}\, f \cdot u = u \cdot f$$

$$\mathsf{T}\, f \cdot \mu = \mu \cdot \mathsf{T}^2\, f$$

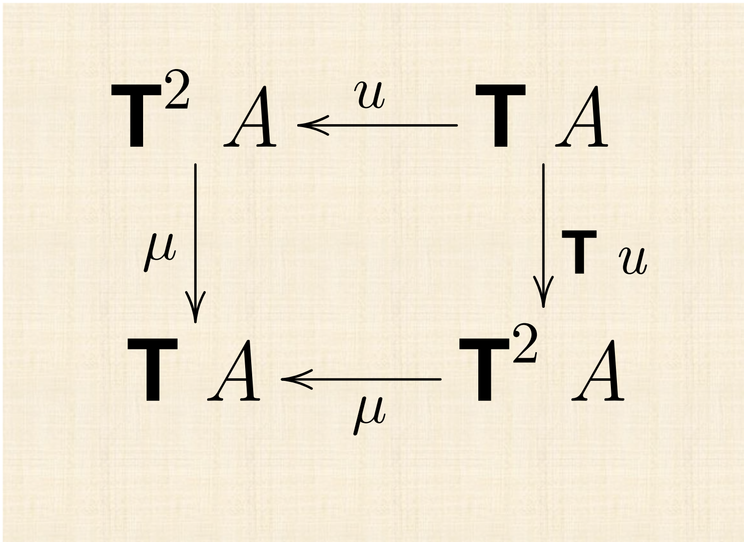
Monad – multiplicação e unidade



Monad – multiplicação e unidade



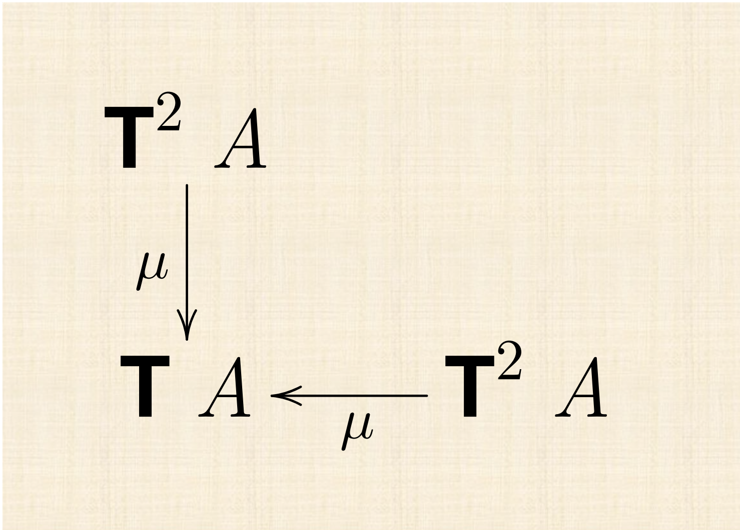
Monad – multiplicação e unidade



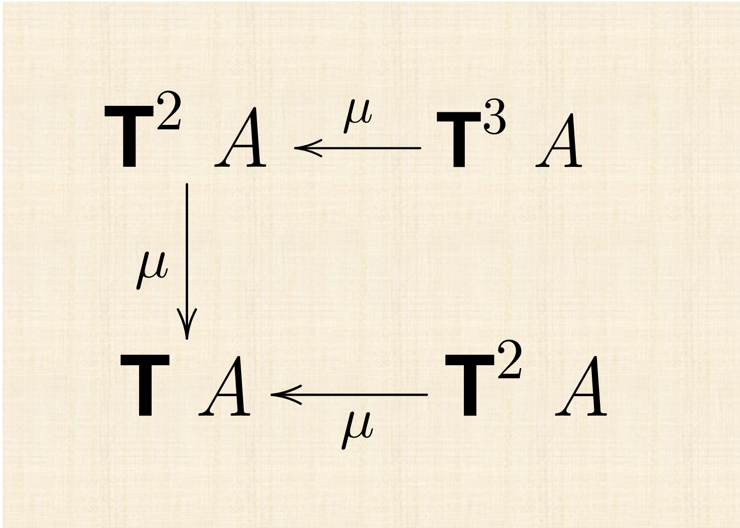
Monad – multiplicação e unidade

$$\mu \cdot u = id = \mu \cdot \mathbf{T}\, u$$

Monad – multiplicação *versus* multiplicação



Monad – multiplicação *versus* multiplicação



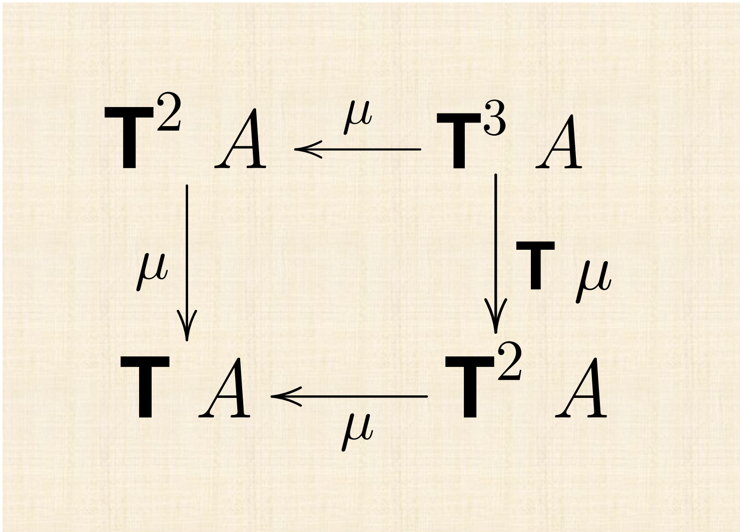
Monad – multiplicação *versus* multiplicação

$$\mathbf{T}\;X \xleftarrow{\;\mu\;} \mathbf{T}^2\;X$$

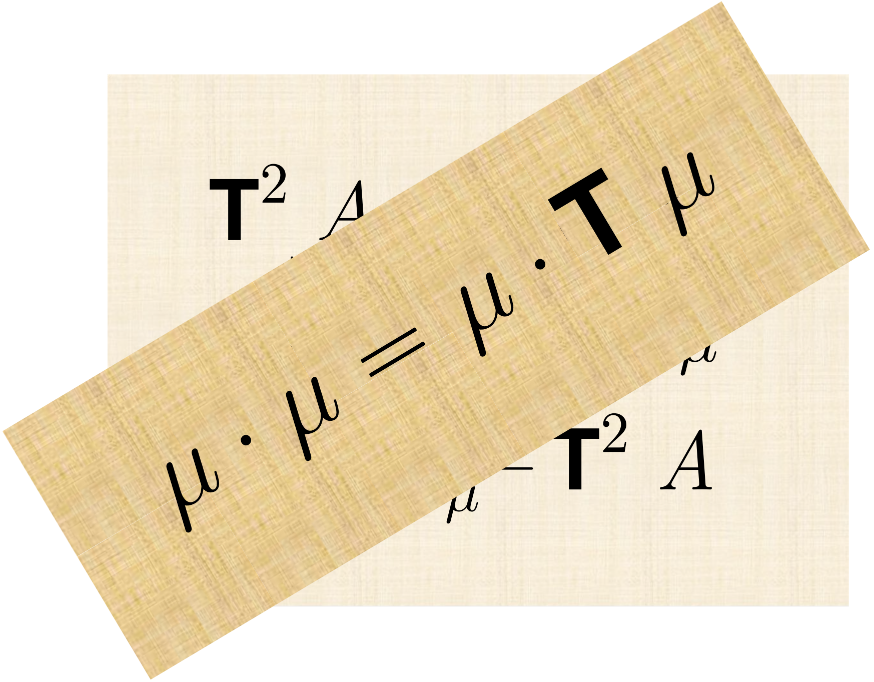
$$\text{where } X = \mathbf{T}\;A$$

$$\begin{array}{ccc}
\mathbf{T}^2\;A & \xleftarrow{\;\mu\;} & \mathbf{T}^3\;A \\
\downarrow{\scriptstyle \mu} & & \\
\mathbf{T}\;A & \xleftarrow[\;\mu\;]{} & \mathbf{T}^2\;A
\end{array}$$

Monad – multiplicação *versus* multiplicação



Monad – multiplicação *versus* multiplicação



Monad – leis da unidade e da [illegible]plicação

$$\mu \cdot u = id = \mu \cdot \mathbf{T}\, u$$

$$\mu \cdot \mu = \mu \cdot \mathbf{T}\, \mu$$

$$X \xrightarrow{\;u\;} \mathbf{T}\, X \xleftarrow{\;\mu\;} \mathbf{T}\,(\mathbf{T}\, X)$$

Monad – leis da unidade e da multiplicação

$$\mu \cdot u = id = \mu \cdot \mathbf{T}\, u$$

$$\mu \cdot \mu = \mu \cdot \mathbf{T}\, \mu$$

Monad – unidade da composição

$$f \bullet u = f = u \bullet f$$

$\equiv$ { definição de $f \bullet g$, duas vezes }

$$\mu \cdot \mathsf{T}\, f \cdot u = f = \mu \cdot \mathsf{T}\, u \cdot f$$

$\equiv$ { natural-u e $\mu \cdot \mathsf{T}\, u = id$ }

$$\mu \cdot u \cdot f = f = id \cdot f$$

$\equiv$ { $\mu \cdot u = id$ }

$$f = f$$

$$\mathsf{T}\, f \cdot u = u \cdot f$$
$$\mathsf{T}\, f \cdot \mu = \mu \cdot \mathsf{T}^2\, f$$

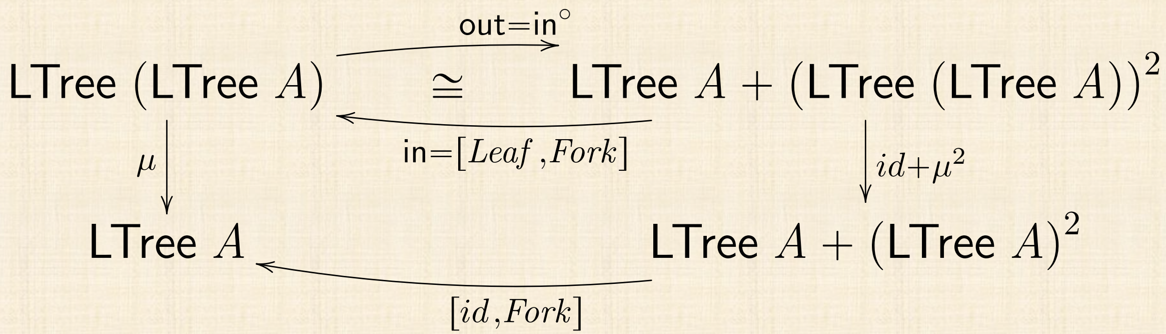
$$\mu \cdot u = id = \mu \cdot \mathsf{T}\, u$$

Monad – leis da unidade e da multiplicação

$$\mu \cdot u = id = \mu \cdot \mathbf{T}\, u$$

$$\mu \cdot \mu = \mu \cdot \mathbf{T}\, \mu$$

Monad LTree



$$\mu = (\!|\, [id, Fork] \,|\!)$$

```
data LTree a = Leaf a | Fork (LTree a, LTree a)
```

```
mu  :: LTree (LTree a) -> LTree a
mu  =  cataLTree (either id Fork)
```

Monad LTree

$$X \xrightarrow{Leaf} \mathsf{LTree}\ X \xleftarrow{(\![id, Fork]\!)} \mathsf{LTree}^2\ X$$

$$\mu \cdot u = id = \mu \cdot \mathbf{T}\ u$$

$$\mu \cdot \mu = \mu \cdot \mathbf{T}\ \mu$$

```
data LTree a = Leaf a | Fork (LTree a, LTree a)
```

```
mu  :: LTree (LTree a) -> LTree a
mu  =  cataLTree (either id Fork)
```

Monad LTree

$$
\begin{array}{ll}
 & \mu \cdot \mathsf{LTree}\ \mathit{Leaf} = \mathit{id} \\
\equiv & \quad \{ \text{ definição de } \mu \ \} \\
 & (\![\, [\mathit{id}, \mathit{Fork}] \,]\!) \cdot \mathsf{LTree}\ \mathit{Leaf} = \mathit{id} \\
\equiv & \quad \{ \text{ absorção-cata } \} \\
 & (\![\, [\mathit{id}, \mathit{Fork}] \cdot (\mathit{Leaf} + \mathit{id}) \,]\!) = \mathit{id} \\
\equiv & \quad \{ \text{ absorção-+ } \} \\
 & (\![\, [\mathit{Leaf}, \mathit{Fork}] \,]\!) = \mathit{id} \\
\equiv & \quad \{ \text{ reflexão-cata } \} \\
 & \mathit{true}
\end{array}
$$

$$\mu \cdot u = id = \mu \cdot \mathbf{T}\ u$$

$$X \xrightarrow{\ \mathit{Leaf}\ } \mathsf{LTree}\ X \xleftarrow{\ (\![\, [\mathit{id}, \mathit{Fork}] \,]\!)\ } \mathsf{LTree}^2\ X$$

```haskell
data LTree a = Leaf a | Fork (LTree a, LTree a)
```

```haskell
mu :: LTree (LTree a) -> LTree a
mu =  cataLTree (either id Fork)
```

Monad "Binding"

$$A \xrightarrow{u} \mathbf{T}\, A \xleftarrow{\mu} \mathbf{T}^2\, A$$

$$f \bullet u = f$$

Monad "Binding"

$$A \xrightarrow{u} \mathbf{T}\, A \xleftarrow{\mu} \mathbf{T}^2\, A$$

$$f \bullet id = ?$$

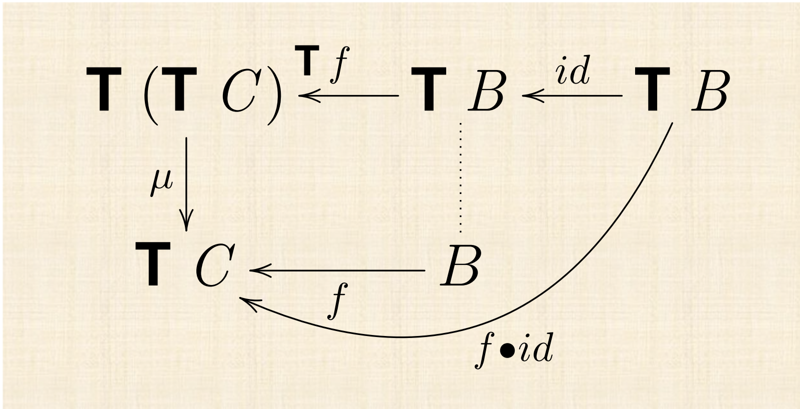
$$f \bullet u = f$$

Monad “Binding”

$$A \xrightarrow{u} \mathbf{T}\,A \xleftarrow{\mu} \mathbf{T}^2\,A$$

$$f \bullet id = ?$$

$$f \bullet u = f$$

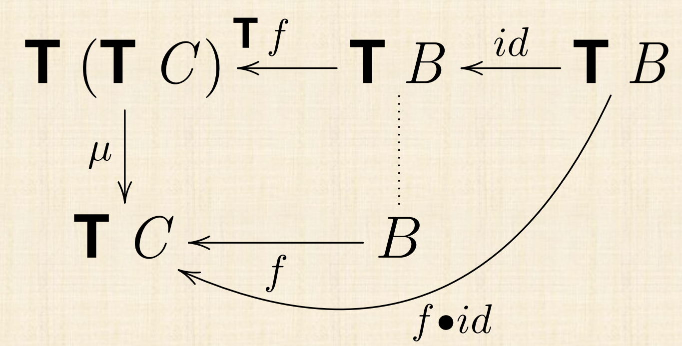


Monad "Binding"

$$A \xrightarrow{u} \mathbf{T}\,A \xleftarrow{\mu} \mathbf{T}^2\,A$$

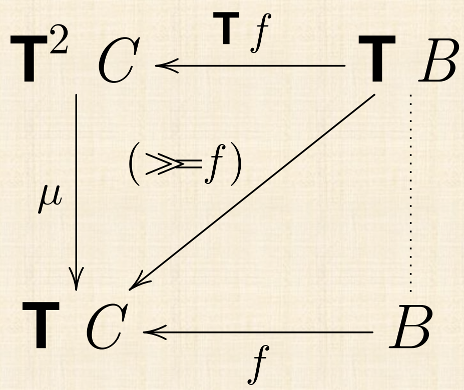
$$f \bullet id = \mu \cdot \mathbf{T}\,f$$

$$f \bullet u = f$$



Monad “Binding”

$$A \xrightarrow{u} \mathbf{T}\ A \xleftarrow{\mu} \mathbf{T}^2\ A$$

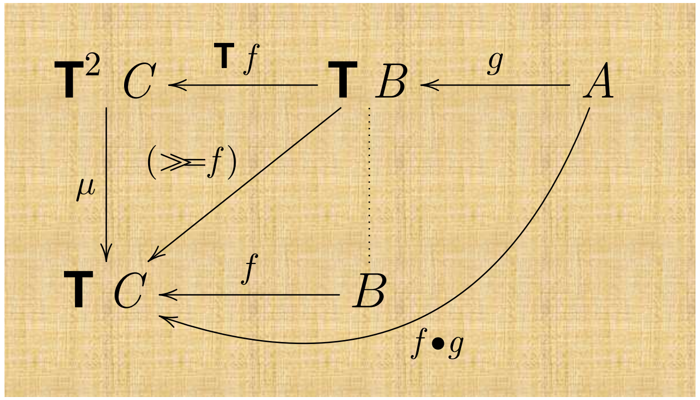


$$(\gg\!= f) = f \bullet id$$

$$x \gg\!= f \overset{\text{def}}{=} (\mu \cdot \mathbf{T}\ f)x$$

Monad “Binding”

$$A \xrightarrow{u} \mathbf{T}\ A \xleftarrow{\mu} \mathbf{T}^2\ A$$



$$(\gg\!\!= f) = f \bullet id$$

$$(f \bullet g)\ x = (g\ x) \gg\!\!= f$$

Monad (Haskell)

```
-- (5) Monad -----------------------

instance Monad LTree where
    return  = Leaf
    t >>= g = (mu . fmap g) t

mu  :: LTree (LTree a) -> LTree a
mu  =  cataLTree (either id Fork)
```

Minimal complete definition

`(>>=)`

Methods

`(>>=) :: forall a b. m a -> (a -> m b) -> m b` infixl 1 # Source

Sequentially compose two actions, passing any value produced by the first as an argument to the second.

`(>>) :: forall a b. m a -> m b -> m b` infixl 1 # Source

Sequentially compose two actions, discarding any value produced by the first, like sequencing operators (such as the semicolon) in imperative languages.

`return :: a -> m a` # Sourc

$$\text{return} = u$$

Monad (Haskell)

```haskell
-- (7) Monads: ---------------------------------------------

-- (7.1) Kleisli monadic composition -----------------------

infix 4  .!

(.!) :: Monad a => (b -> a c) -> (d -> a b) -> d -> a c
(f .! g) a = (g a) >>= f

mult :: (Monad m) => m (m b) -> m b
mult = (>>= id)  -- also known as join
```

Minimal complete definition

`(>>=)`

Methods

`(>>=) :: forall a b. m` # Source

Sequentially compose two actions, passing any value produced by the first as an argument to the second.

$$(f \bullet g)\ x = (g\ x) \gg= f$$

`(>>) :: forall a b. m a -> m b -> m b` infixl 1 # Source

Sequentially compose two actions, discarding any value produced by the first, like sequencing operators (such as the semicolon) in imperative languages.

$$id \bullet id \quad = \quad \mu$$

`return :: a -> m a` # Source

$$\text{return} = u$$

Monad (Haskell) IO

Minimal complete definition

```
(>>=)
```

Methods

```
(>>=) :: forall a b. m a -> (a -> m b) -> m b    infixl 1   # Source
```

Sequentially compose two actions, passing any value produced by the first as an argument to the second.

```
(>>) :: forall a b. m a -> m b -> m b    infixl 1   # Source
```

Sequentially compose two actions, discarding any value produced by the first, like sequencing operators (such as the semicolon) in imperative languages.

```
return :: a -> m a    # Sourc
```

$\text{return} = u$

Monad (Haskell)

```haskell
-- (5) Monad -----------------------

instance Monad LTree where
    return  = Leaf
    t >>= g = (mu . fmap g) t

mu  :: LTree (LTree a) -> LTree a
mu  =  cataLTree (either id Fork)
```

Minimal complete definition

`(>>=)`

Methods

`(>>=) :: forall a b. m a -> (a -> m b) -> m b` | infixl 1 | # Source

Sequentially compose two actions, passing any value produced by the first as an argument to the second.

`(>>) :: forall a b. m a -> m b -> m b` | infixl 1 | # Source

Sequentially compose two actions, discarding any value produced by the first, like sequencing operators (such as the semicolon) in imperative languages.

`return :: a -> m a` | # Sourc

$$\text{return} = u$$

MONAD IDENTIDADE

$$\mathbf{T}\ X = X$$
$$\mathbf{T}\ f = f$$

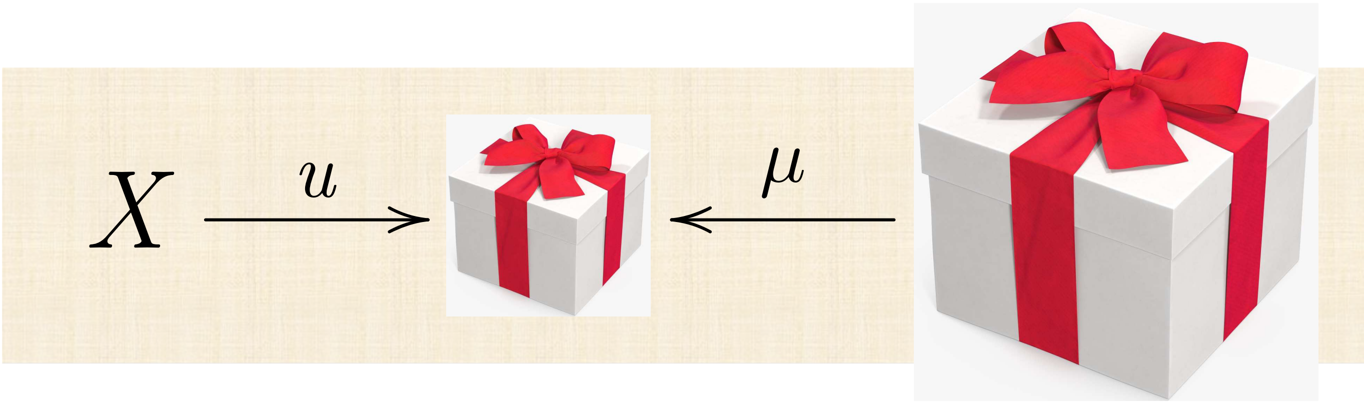
$$X \xrightarrow{u} \mathbf{T}\ X \xleftarrow{\mu} \mathbf{T}\ (\mathbf{T}\ X)$$

$$X \xrightarrow{u} X \xleftarrow{\mu} X$$

$$u = id$$
$$\mu = id$$
$$f \bullet g = \mu \cdot \mathbf{T}\ f \cdot g = id \cdot f \cdot g = f \cdot g$$

Cálculo de Programas
Aula T12

MONAD - recordar

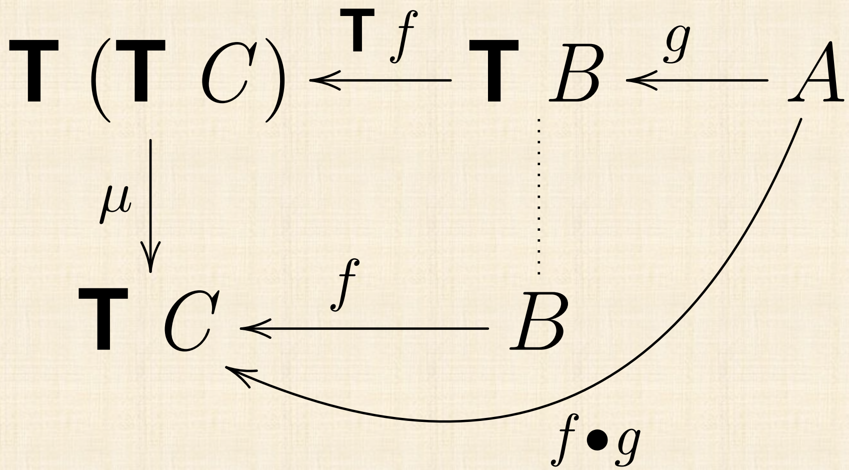


Monad =
functor "de
corridas"

$$X \xrightarrow{u} \mathbf{T}\, X \xleftarrow{\mu} \mathbf{T}\,(\mathbf{T}\, X)$$

Composição monádica

$$X \xrightarrow{u} \mathbf{T}\,X \xleftarrow{\mu} \mathbf{T}\,(\mathbf{T}\,X)$$



Monads – sumário (leis e binding)

$$\mu \cdot u = id = \mu \cdot \mathsf{T}\ u$$

$$\mu \cdot \mu = \mu \cdot \mathsf{T}\ \mu$$

$$(\gg\!= f) = f \bullet id$$

MONADS: notação-do

$$X \xrightarrow{u} \mathsf{T}\, X \xleftarrow{\mu} \mathsf{T}\,(\mathsf{T}\, X)$$

$$(f \cdot g)\ a = f\ (g\ a) = \mathbf{let}\ b = g\ a\ \mathbf{in}\ f\ b$$

$$(f \bullet g)\ a = \mathbf{do}\ \{ b \leftarrow g\ a; f\ b \}$$

MONADS: notação-do

$$X \xrightarrow{u} \mathbf{T}\ X \xleftarrow{\mu} \mathbf{T}\ (\mathbf{T}\ X)$$

$$(f \cdot g)\ a = f\ (g\ a) = \mathbf{let}\ b = g\ a\ \mathbf{in}\ f\ b$$

$$(f \bullet g)\ a = \mathbf{do}\ \{ b \leftarrow \ldots f\ b \}$$

MONADS: notação-do

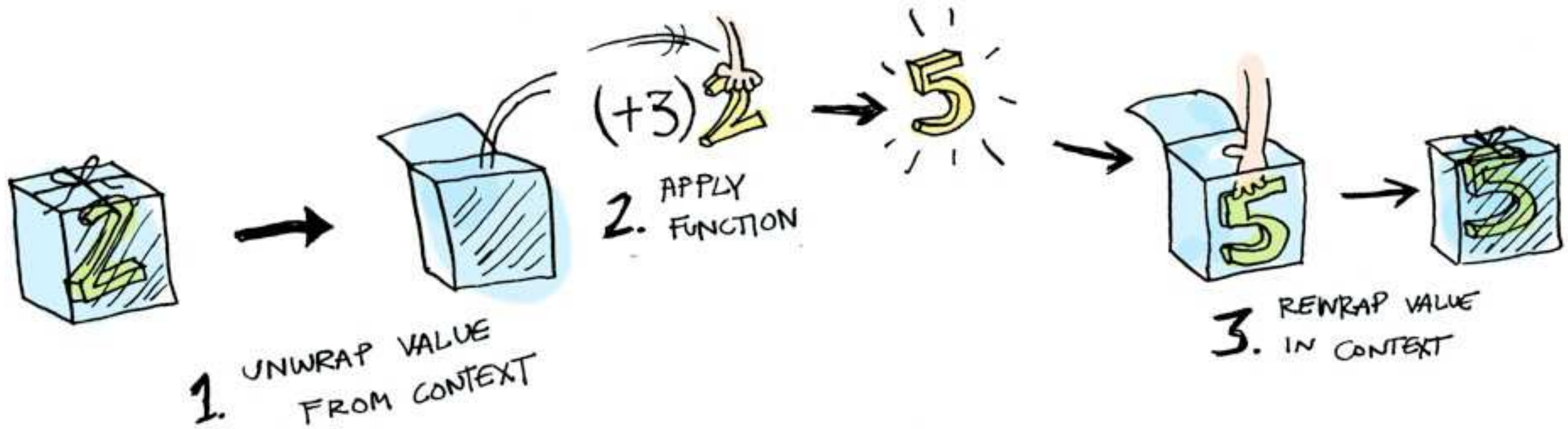
$$x \gg= f = \mathbf{do}\ \{b \leftarrow x; f\ b\}$$

$$
\begin{array}{ll}
 & x \gg= f \\
= & \{ \text{ definição de } (\gg= f) \ \} \\
 & (f \bullet id)\ x \\
= & \{ \text{ notação-}\mathbf{do} \ \} \\
 & \mathbf{do}\ \{b \leftarrow x; f\ b\}
\end{array}
$$

$$(f \bullet g)\ a = \mathbf{do}\ \{b \leftarrow g\ a; f\ b\}$$

MONADS: notação-do

$$x \gg= f = \mathbf{do}\ \{\, b \leftarrow x; f\ b \,\}$$



(**Créditos**: `http://shorturl.at/buNPX`)

MONADS: notação-do

$$(g \cdot f) \bullet h = g \bullet (\mathsf{T}\ f \cdot h)$$

Caso particular: $g, h := u, id$:

$$(u \cdot f) \bullet id = u \bullet (\mathsf{T}\ f \cdot id)$$

$\equiv$ { natural-id e unidade u }

$$(u \cdot f) \bullet id = \mathsf{T}\ f$$

$\equiv$ { passagem a *pointwise* }

$$((u \cdot f) \bullet id)\ x = \mathsf{T}\ f\ x$$

$\equiv$ { passagem para notação-**do** }

$$\mathsf{T}\ f\ x = \mathbf{do}\ \{b \leftarrow x; \mathsf{return}\ (f\ b)\}$$

$$(f \bullet g)\ a = \mathbf{do}\ \{b \leftarrow g\ a; f\ b\}$$

MONADS: leis em notação-do

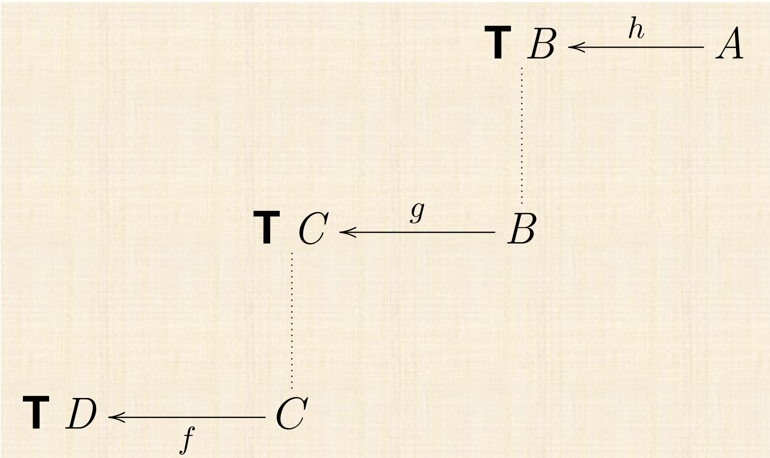
$$u \bullet f = f = f \bullet u$$

$$\begin{array}{ll} & u \bullet f = f = f \bullet u \\ \equiv & \{ \text{ passagem a } \textit{pointwise} \ \} \\ & (u \bullet f)\ a = f\ a = (f \bullet u)\ a \\ \equiv & \{ \text{ notação-}\mathbf{do} \ \} \\ & \mathbf{do}\ \{ b \leftarrow f\ a; \mathsf{return}\ b \} = f\ a = \mathbf{do}\ \{ b \leftarrow \mathsf{return}\ a; f\ b \} \end{array}$$

$$(f \bullet g)\ a = \mathbf{do}\ \{ b \leftarrow g\ a; f\ b \}$$

MONADS: leis vertidas para notação-do

$$f \bullet (g \bullet h) = (f \bullet g) \bullet h$$



$$(f \bullet g)\ a = \mathbf{do}\ \{ b \leftarrow g\ a; f\ b \}$$

MONADS: leis vertidas para notação-do

$$f \bullet (g \bullet h) = (f \bullet g) \bullet h$$

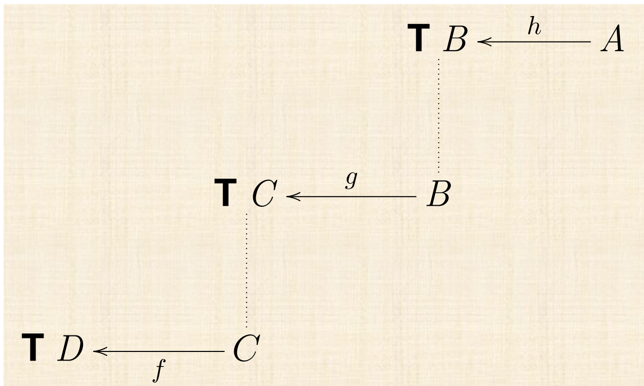
$$(f \bullet (g \bullet h))\ a$$

$\equiv$ { notação-do } $\qquad (f \bullet g)\ a = \mathbf{do}\ \{ b \leftarrow g\ a; f\ b \}$

$$\mathbf{do}\ \{ c \leftarrow (g \bullet h)\ a; f\ c \}$$

$\equiv$ { notação-do } $\qquad (f \bullet g)\ a = \mathbf{do}\ \{ b \leftarrow g\ a; f\ b \}$

$$\mathbf{do}\ \{ c \leftarrow \mathbf{do}\ \{ b \leftarrow h\ a; g\ b \}; f\ c \}$$



MONADS: leis vertidas para notação-do

$$f \bullet (g \bullet h) = (f \bullet g) \bullet h$$

$$((f \bullet g) \bullet h)\ a$$

$\equiv$ { notação-do } $\quad (f \bullet g)\ a = \textbf{do}\ \{b \leftarrow g\ a; f\ b\}$

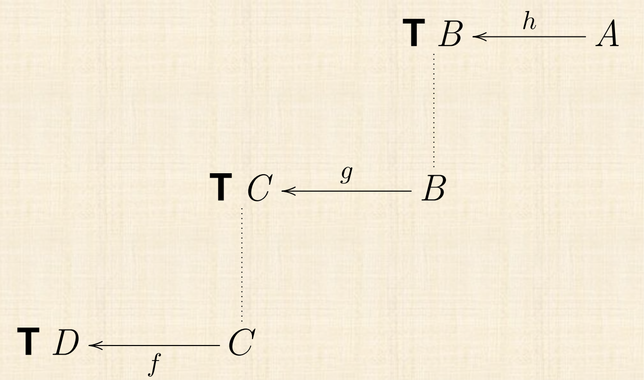
$$\textbf{do}\ \{b \leftarrow h\ a; (f \bullet g)\ b\}$$

$\equiv$ { notação-do } $\quad (f \bullet g)\ a = \textbf{do}\ \{b \leftarrow g\ a; f\ b\}$

$$\textbf{do}\ \{b \leftarrow h\ a; \textbf{do}\ \{c \leftarrow g\ b; f\ c\}\}$$

$\equiv$ { simplificação }

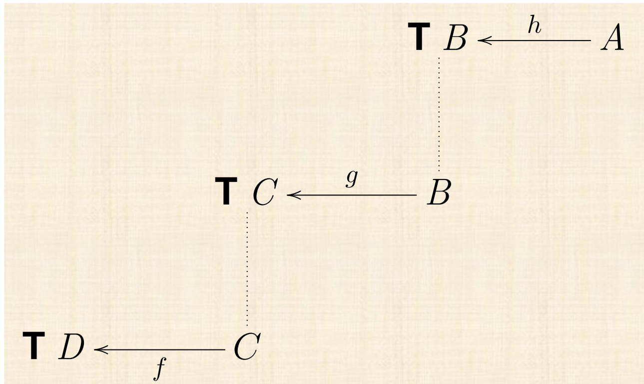
$$\textbf{do}\ \{b \leftarrow h\ a; c \leftarrow g\ b; f\ c\}$$



MONADS: leis vertidas para notação-do

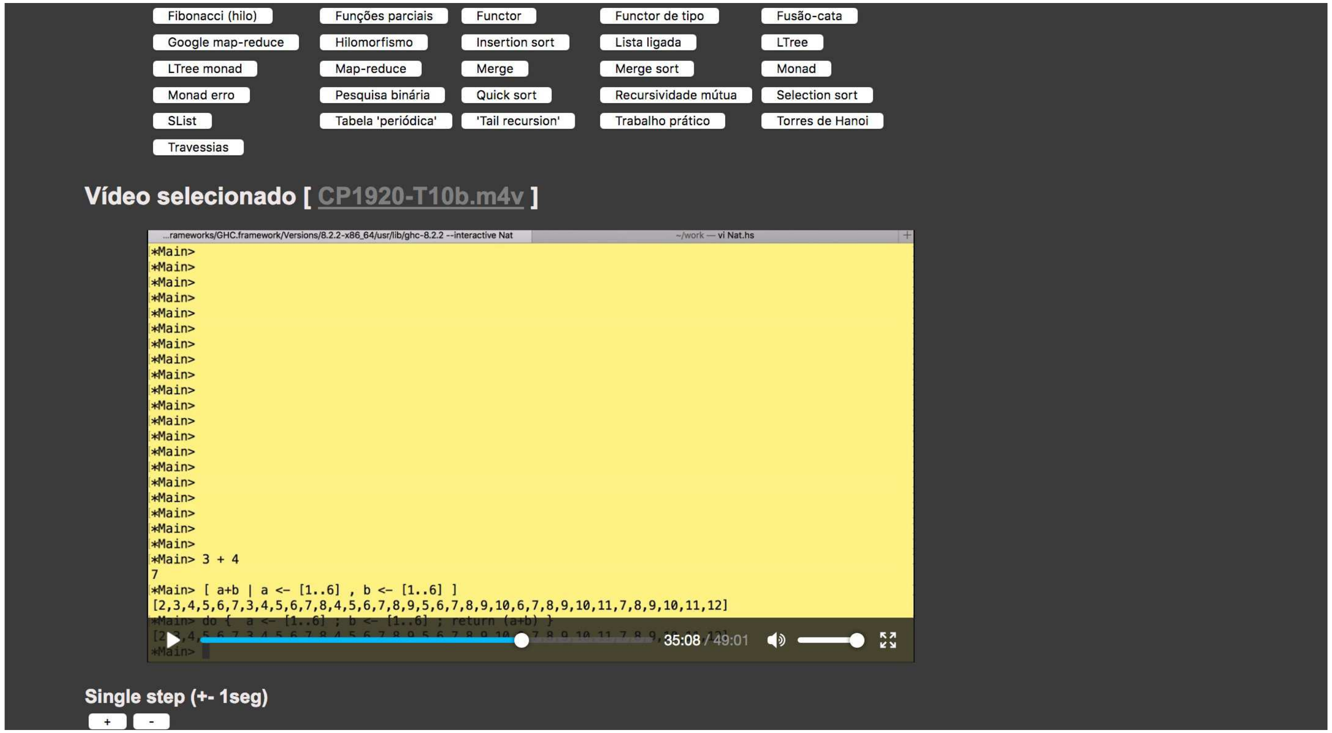
$$f \bullet (g \bullet h) = (f \bullet g) \bullet h$$

$$\mathbf{do}\ \{c \leftarrow \mathbf{do}\ \{b \leftarrow h\ a; g\ b\}; f\ c\} = \mathbf{do}\ \{b \leftarrow h\ a; c \leftarrow g\ b; f\ c\}$$



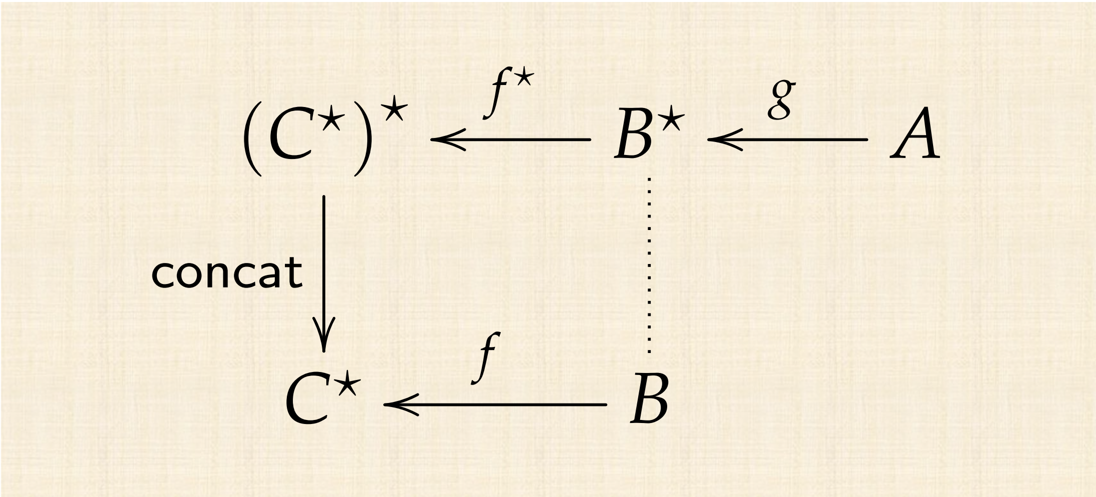
$$(f \bullet g)\ a = \mathbf{do}\ \{b \leftarrow g\ a; f\ b\}$$

MONADS: listas em compreensão



MONADS: listas em compreensão

$$[\, e \mid a_1 \leftarrow x_1, \ldots, a_n \leftarrow x_n \,] = \textbf{do}\ \{ a_1 \leftarrow x_1; \ldots; a_n \leftarrow x_n; \mathsf{return}\ e \}$$



Formulário

Mónadas

Multiplicação	$\mu \cdot \mu = \mu \cdot \mathsf{T}\,\mu$	(61)
Unidade	$\mu \cdot u = \mu \cdot \mathsf{T}\,u = id$	(62)
Natural-u	$u \cdot f \quad = \quad \mathsf{T}\,f \cdot u$	(63)
Natural-μ	$\mu \cdot \mathsf{T}\,(\mathsf{T}\,f) \quad = \quad \mathsf{T}\,f \cdot \mu$	(64)
Composição monádica	$f \bullet g \quad = \quad \mu \cdot \mathsf{T}\,f \cdot g$	(65)
Associatividade-$\bullet$	$f \bullet (g \bullet h) = (f \bullet g) \bullet h$	(66)
Identidade-$\bullet$	$u \bullet f = f = f \bullet u$	(67)
Associatividade-$\bullet/\cdot$	$(f \bullet g) \cdot h \quad = \quad f \bullet (g \cdot h)$	(68)
Associatividade-$\cdot/\bullet$	$(f \cdot g) \bullet h \quad = \quad f \bullet (\mathsf{T}\,g \cdot h)$	(69)
μ **versus** $\bullet$	$id \bullet id \quad = \quad \mu$	(70)

Formulário

Composição monádica	$(f \bullet g)\ a = \mathbf{do}\ \{\, b \leftarrow g\ \ a; f\ \ b\,\}$	(85)
'Binding as μ'	$x \ggg f \quad = \quad (\mu \cdot \mathsf{T}\ f)x$	(86)
Notação-do	$\mathbf{do}\ \{\, x \leftarrow a; b\,\} \quad = \quad a \ggg (\lambda x \rightarrow b)$	(87)
'μ as binding'	$\mu\, x \quad = \quad x \ggg id$	(88)
Sequenciação	$x \gg y \quad = \quad x \ggg \underline{y}$	(89)

MONAD I/O: interface com sistema de ficheiros

MONAD I/O: interface com sistema de ficheiros

$$
\begin{array}{ccccc}
 & & \mathsf{IO}\ String & \xleftarrow{\ readFile\ } & FilePath \\
 & & \vdots & & \\
\mathsf{IO}\ 1 & \xleftarrow{\ writeFile\ o\ } & String & &
\end{array}
$$

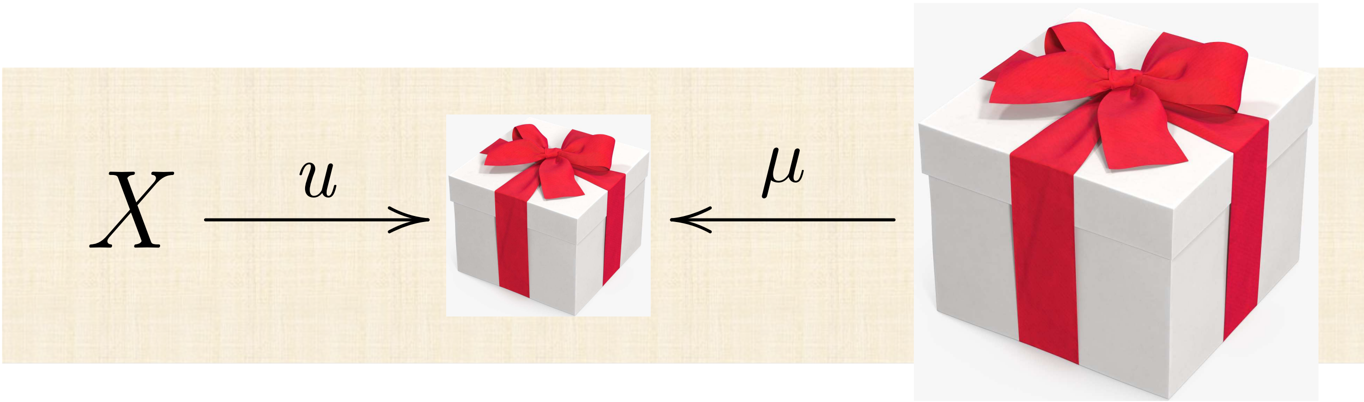
$$copy\ i\ o = (writeFile\ o \bullet readFile)\ i$$

$$\equiv \qquad \{\ \text{notação-do}\ \}$$

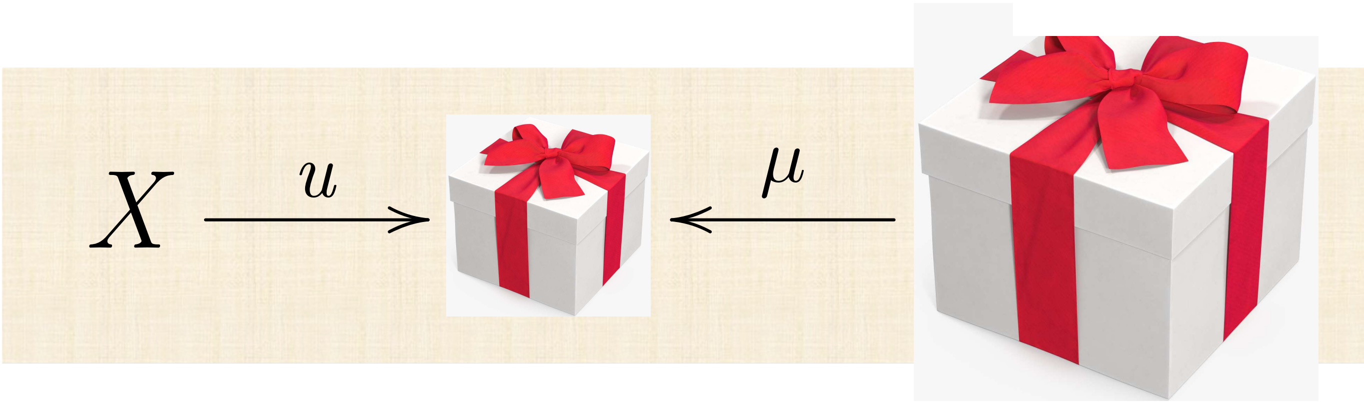
$$copy\ i\ o = \mathbf{do}\ \{ s \leftarrow readFile\ i;\ writeFile\ o\ s \}$$

O monad (trivial) da identidade.

MONADS binding



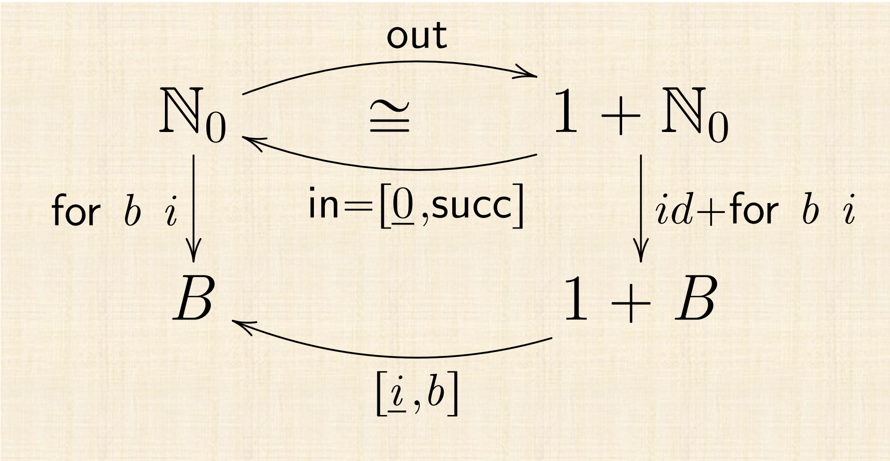
MONADS (recordar)



Programação monádica **probabilística**

RECURSIVIDADE
monádica

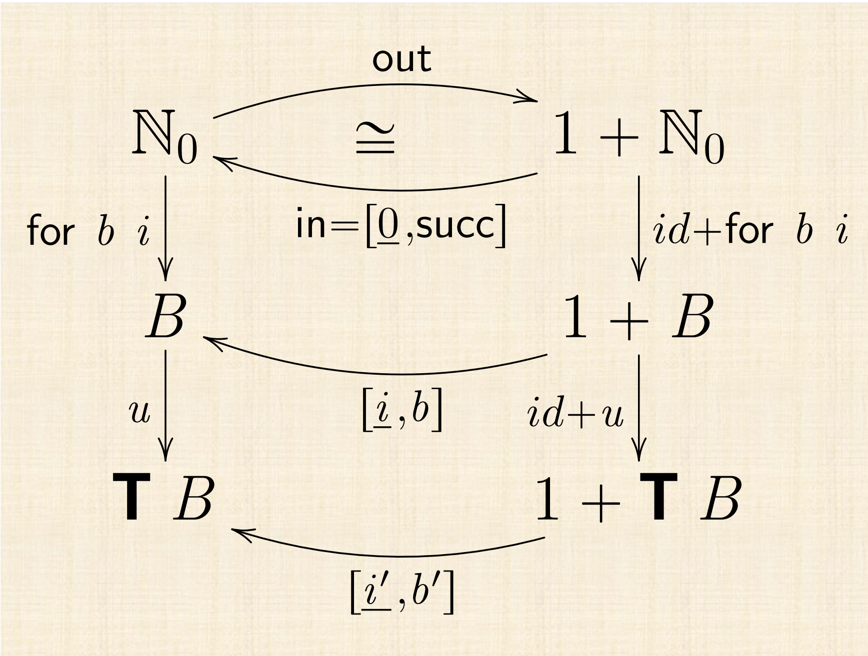
$$X \xrightarrow{u} \mathsf{T}\,X \xleftarrow{\mu} \mathsf{T}\,(\mathsf{T}\,X)$$



$$\mathsf{for}\ b\ i = (\![\,[\underline{i}, b]\,]\!)$$

RECURSIVIDADE
monádica

$$X \xrightarrow{\;u\;} \mathbf{T}\, X \xleftarrow{\;\mu\;} \mathbf{T}\,(\mathbf{T}\, X)$$



for b $i = (\!|\, [\underline{i}, b] \,|\!)$

RECURSIVIDADE
monádica

$$X \xrightarrow{u} \mathsf{T}\ X \xleftarrow{\mu} \mathsf{T}\ (\mathsf{T}\ X)$$

$$
\begin{array}{ll}
 & u \cdot \mathsf{for}\ b\ i = \mathsf{for}\ b'\ i' \\
\equiv & \quad \{ \ \mathsf{for}\ b\ i = (\![[\underline{i}, b]]\!) \ \} \\
 & u \cdot (\![[\underline{i}, b]]\!) = (\![[\underline{i'}, b']]\!) \\
\Leftarrow & \quad \{ \ \text{fusão-cata} \ \} \\
 & u \cdot [\underline{i}, b] = [\underline{i'}, b'] \cdot (id + u) \\
\equiv & \quad \{ \ \text{coprodutos (fusão, absorção, eq)} \ \} \\
 & \left\{ \begin{array}{l} u \cdot \underline{i} = \underline{i'} \\ u \cdot b = b' \cdot u \end{array} \right.
\end{array}
$$

RECURSIVIDADE

monádica

$$X \xrightarrow{\ u\ } \mathbf{T}\ X \xleftarrow{\ \mu\ } \mathbf{T}\ (\mathbf{T}\ X)$$

$$u \cdot \mathsf{for}\ b\ i = \mathsf{for}\ b'\ i'$$

$\equiv$ { $\mathsf{for}\ b\ i = (\!|\,[\underline{i}, b]\,|\!)$ }

$$u \cdot (\!|\,[\underline{i}, b]\,|\!) = (\!|\,[\underline{i'}, b']\,|\!)$$

$\Leftarrow$ { fusão-cata }

$$u \cdot [\underline{i}, b] = [\underline{i'}, b'] \cdot (id + u)$$

$\equiv$ { coprodutos (fusão, absorção, eq) }

$$\begin{cases} u \cdot \underline{i} = \underline{i'} \\ u \cdot b = b' \cdot u \end{cases}$$

$\equiv$ { $f \cdot \underline{x} = \underline{f\ x}$; $\mathbf{T}\ f \cdot u = u \cdot f$ }

$$\begin{cases} u\ i = i' \\ \mathbf{T}\ b \cdot u = b' \cdot u \end{cases}$$

$\Leftarrow$ { trivial }

$$\begin{cases} i' = u\ i \\ b' = \mathbf{T}\ b \end{cases}$$

$$\mathsf{mfor}\ b\ i = (\!|\,[\underline{u\ i}, \mathbf{T}\ b]\,|\!)$$

RECURSIVIDADE
monádica

$$X \xrightarrow{u} \mathbf{T}\ X \xleftarrow{\mu} \mathbf{T}\ (\mathbf{T}\ X)$$

$$\mathsf{mfor}\ b\ i = (\![\, \underline{u\ i}, \mathbf{T}\ b \,]\!)$$

$$\mathsf{mfor}\ b\ i = (\![\, \underline{u\ i}, \mathbf{T}\ b \,]\!)$$

$$\equiv \quad \{ \text{ universal cata } \}$$

$$\mathsf{mfor}\ b\ i \cdot [\underline{0}, \mathsf{succ}] = [\underline{u\ i}, \mathbf{T}\ b] \cdot (id + \mathsf{mfor}\ b\ i)$$

$$\equiv \quad \{ \text{ coprodutos (fusão, absorção, eq) ; variáveis } \}$$

$$\begin{cases} \mathsf{mfor}\ b\ i\ 0 = u\ i \\ \mathsf{mfor}\ b\ i\ (n+1) = \mathbf{T}\ b\ (\mathsf{mfor}\ b\ i\ n) \end{cases}$$

$$\equiv \quad \{ \ u = \mathsf{return}\ ;\ \mathbf{T}\ f\ x = \mathbf{do}\ \{a \leftarrow x; \mathsf{return}\ (f\ a)\} \ \}$$

$$\begin{cases} \mathsf{mfor}\ b\ i\ 0 = \mathsf{return}\ i \\ \mathsf{mfor}\ b\ i\ (n+1) = \mathbf{do}\ \{x \leftarrow \mathsf{mfor}\ b\ i\ n; \mathsf{return}\ (b\ x)\} \end{cases}$$

RECURSIVIDADE

monádica

$$X \xrightarrow{u} \mathbf{T}\, X \xleftarrow{\mu} \mathbf{T}\,(\mathbf{T}\, X)$$

```
mfor b i 0 = i
mfor b i (n+1) = do { x <- mfor b i n ; b x }
```

RECURSIVIDADE

monádica

$$X \xrightarrow{u} \mathbf{T}\ X \xleftarrow{\mu} \mathbf{T}\ (\mathbf{T}\ X)$$

```
mfor b i 0 = i
mfor b i (n+1) = do { x <- mfor b i n ; b x }
```

$$(f \bullet g)\ a = \mathbf{do}\ \{b \leftarrow g\ a; f\ b\}$$

$$
\begin{array}{ll}
 & (b \bullet (\textsf{mfor}\ b\ i))\ n \\
= & \quad \{ \text{ composição e cxomposição monádica } \} \\
 & (b \bullet id)\ (\textsf{mfor}\ b\ i\ n)
\end{array}
$$

RECURSIVIDADE
monádica

$$X \xrightarrow{u} \mathsf{T}\ X \xleftarrow{\mu} \mathsf{T}\ (\mathsf{T}\ X)$$

```
mfor b i 0 = i
mfor b i (n+1) = do { x <- mfor b i n ; b x }
```

$$(f \bullet g)\ a = \mathbf{do}\ \{b \leftarrow g\ a; f\ b\}$$

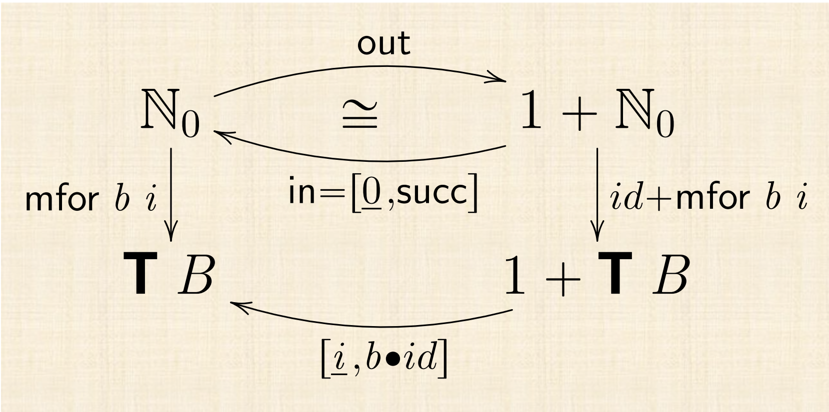
Associatividade-$\bullet$/$\cdot$ $\qquad (f \bullet g) \cdot h = f \bullet (g \cdot h) \qquad (66)$

$$\begin{array}{l} \quad (b \bullet (\mathsf{mfor}\ b\ i))\ n \\ = \qquad \{ \text{ composição e cxomposição monádica } \} \\ \quad (b \bullet id)\ (\mathsf{mfor}\ b\ i\ n) \end{array}$$

RECURSIVIDADE
monádica

$$X \xrightarrow{u} \mathbf{T}\,X \xleftarrow{\mu} \mathbf{T}\,(\mathbf{T}\,X)$$

```
mfor b i 0 = i
mfor b i (n+1) = do { x <- mfor b i n ; b x }
```

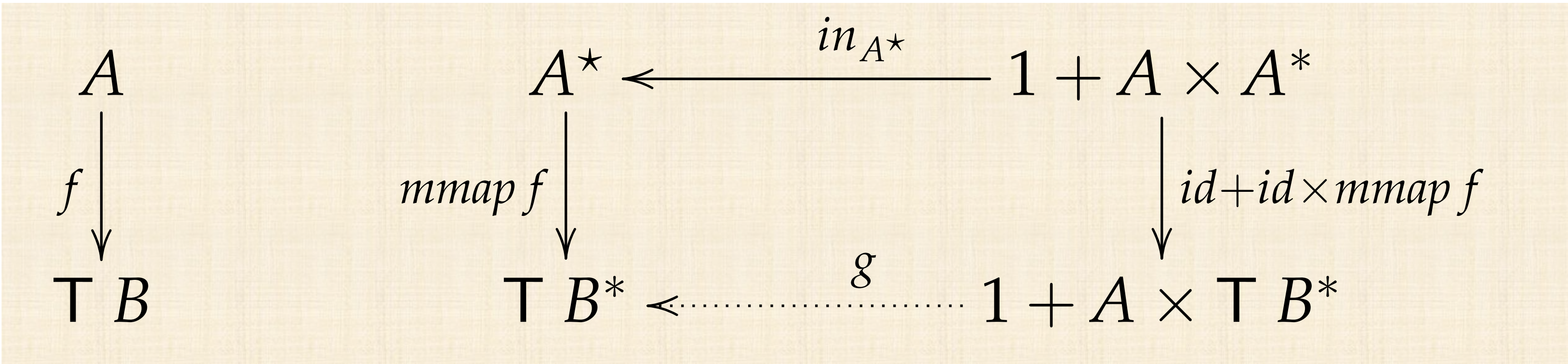


$$1 \xrightarrow{\underline{i}} \mathbf{T}\,B \xleftarrow{b} B$$

$$\mathsf{mfor}\ b\ i = (\![\,[\underline{i}, b \bullet id]\,]\!)$$

RECURSIVIDADE monádica

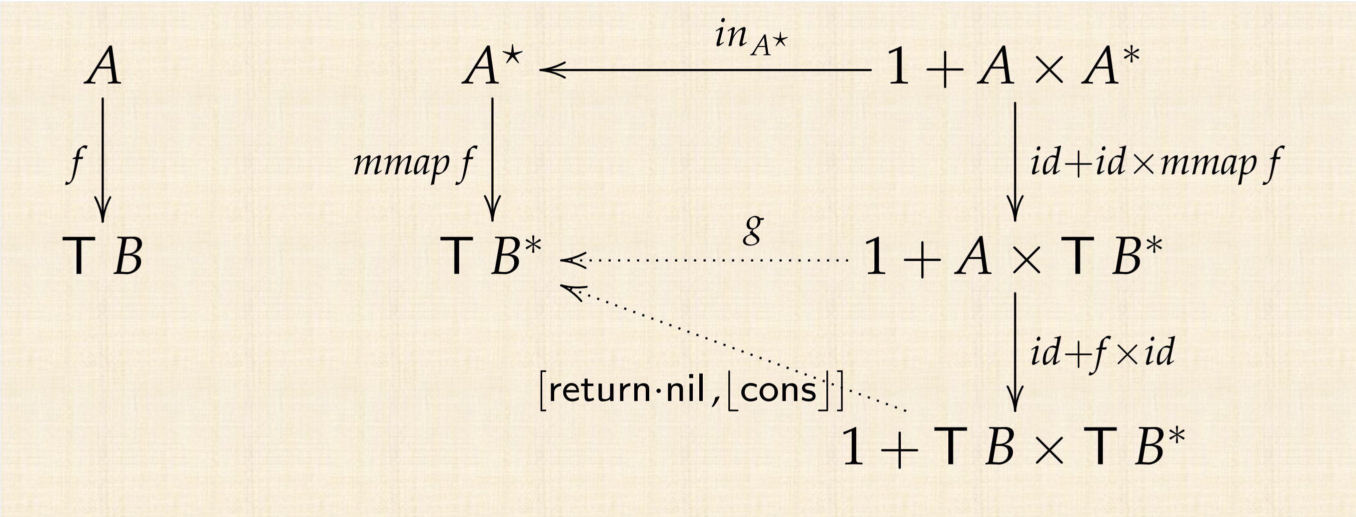
map monádico (mmap)



$g?$

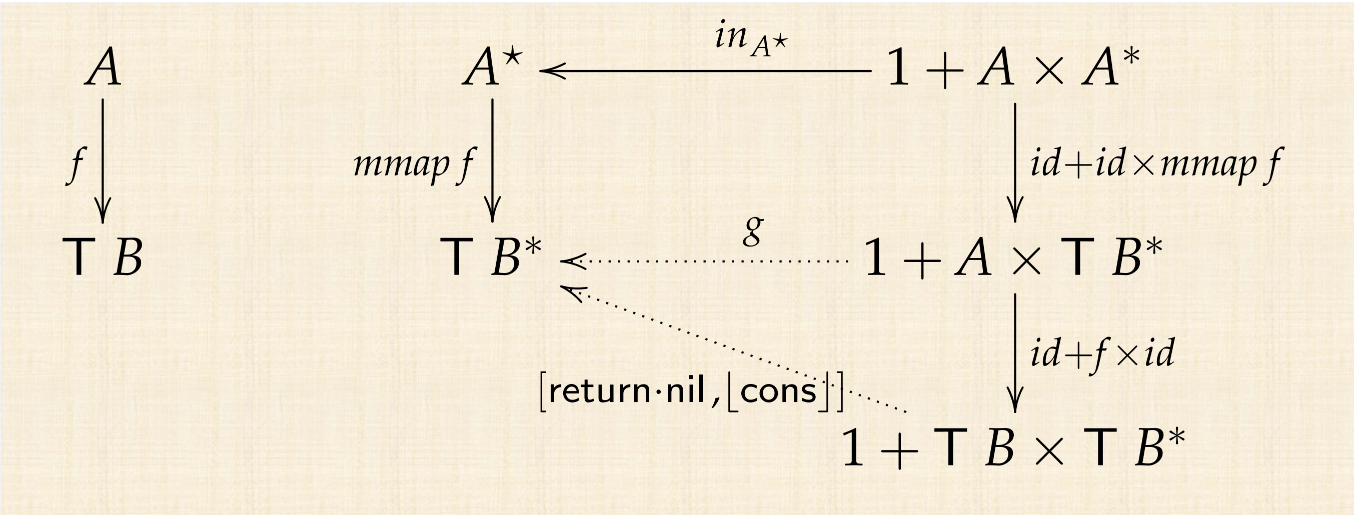
RECURSIVIDADE monádica

map monádico (mmap)



RECURSIVIDADE monádica

map monádico (mmap)



$$\lfloor f \rfloor \; (x, y) = \mathbf{do}\ \{a \leftarrow x; b \leftarrow y; \mathsf{return}\ (f\ (a, b))\}$$

RECURSIVIDADE monádica

map monádico (mmap)

$$mmap :: (Monad\ m) \Rightarrow (a \rightarrow m\ b) \rightarrow [\,a\,] \rightarrow m\,[\,b\,]$$
$$mmap\ f\ [\,] = \mathsf{return}\ [\,]$$
$$mmap\ f\ (h : t) = \mathbf{do}\ \{\,b \leftarrow f\ h; x \leftarrow mmap\ f\ t; \mathsf{return}\ (b : x)\}$$

RECURSIVIDADE monádica

Exemplo (mmap)

Obter o mínimo de uma lista (se possível...):

$$mgetmin :: Ord\ a \Rightarrow [\,a\,] \rightarrow Maybe\ a$$

Obter a lista de mínimos de uma lista de listas
(onde isso for possível):

```
mmap mgetmin [[1,2],[3]] = Just [1,3]
mmap mgetmin [[1,2],[]] = Nothing
```

RECURSIVIDADE monádica

Exemplo (mmap)

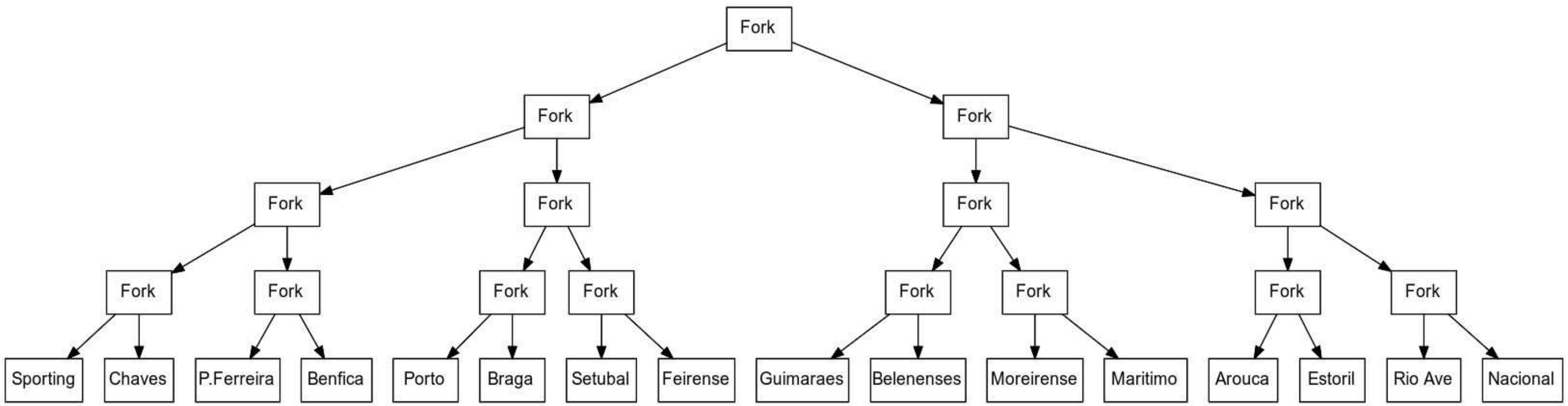
Getting the mimimum of a list (if possible):

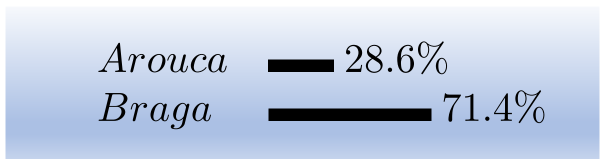
$$mgetmin :: Ord\ a \Rightarrow [\,a\,] \rightarrow Maybe\ a$$

$$\begin{aligned}
&mgetmin\ [\,] = \mathsf{Nothing}\\
&mgetmin\ [\,a\,] = \mathsf{return}\ a\\
&mgetmin\ (h : t) = \mathbf{do}\ \{x \leftarrow mgetmin\ t; \mathsf{return}\ (min\ h\ x)\}
\end{aligned}$$


programming
productivity
work
impetus
compilation
standardized
productivity
tackled
giants
manpower
prototyping
troubleshooters
categorical
hardware
established
estimation
potpourri
methodologies
heyday
project
staffs
smaller
programmer
sigma
shortened
india
software
viewed
techniques
waned
productive
industry
tools
interval
waxed
timelines
iterative
management
promulgated

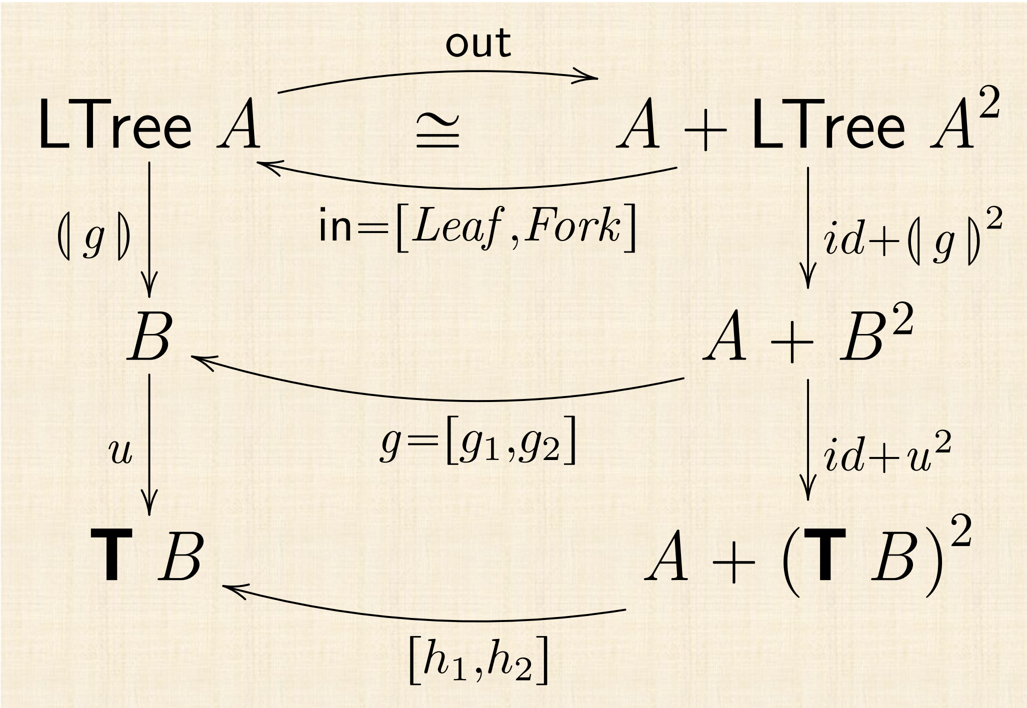
Exemplo – liga de futebol





etc

LTree ... catas com mónades



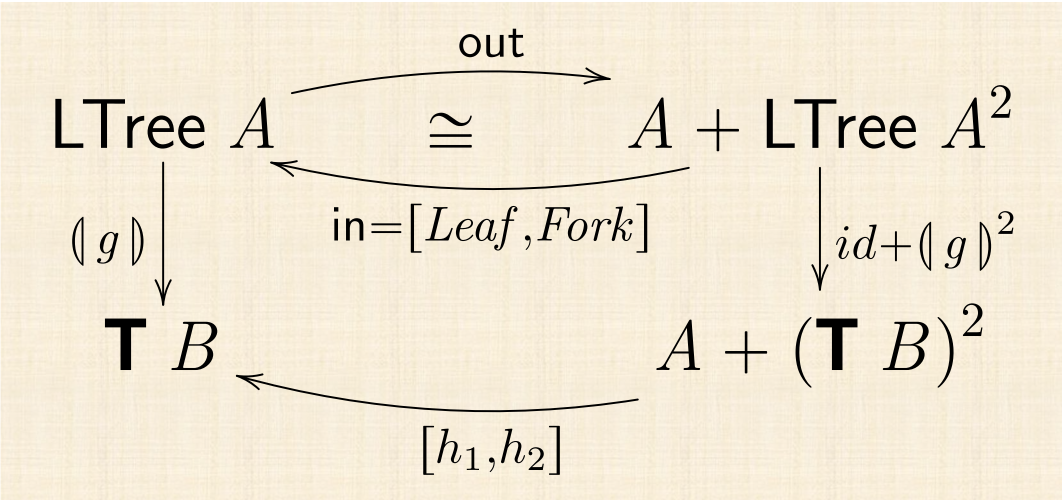
$$\begin{cases} h_1 = u \cdot g_1 \\ h_2 = \mathbf{T}\ g_2 \cdot \delta \end{cases}$$

$\delta\ (x, y) = \mathbf{do}\ \{$
$\quad a \leftarrow x;$
$\quad b \leftarrow y;$
$\quad \mathsf{return}\ (a, b)$
$\quad \}$

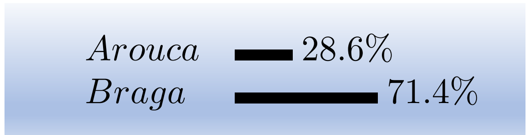
LTree … catas com mónades

$$
\begin{array}{l}
mcataLTtree\ g = k\ \mathbf{where} \\
\quad k\ (Leaf\ \ a) = \mathsf{return}\ (g_1\ a) \\
\quad k\ (Fork\ (x, y)) = \mathbf{do}\ \{\, a \leftarrow k\ x; b \leftarrow k\ y; \mathsf{return}\ (g_2\ (a, b))\,\} \\
\quad g_1 = g \cdot i_1 \\
\quad g_2 = g \cdot i_2
\end{array}
$$

LTree … catas monádicos em geral



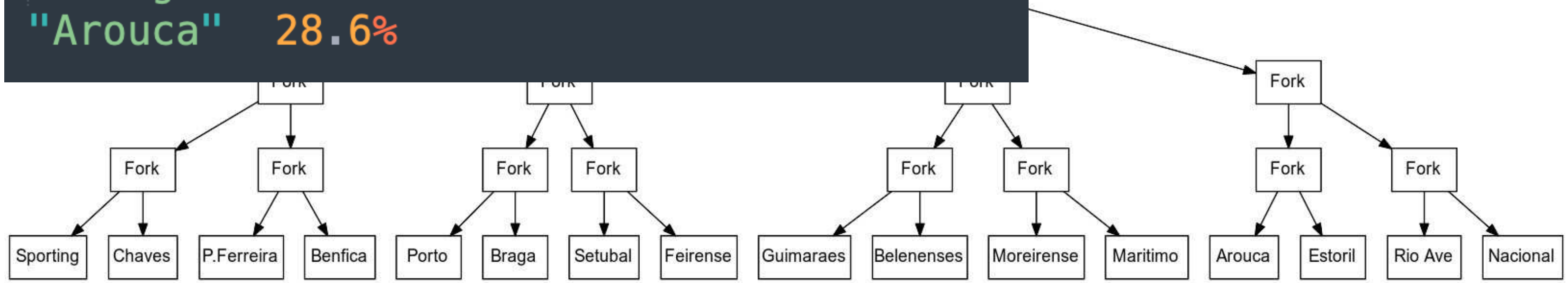
$$f : B^2 \to \mathbf{T}\ B$$

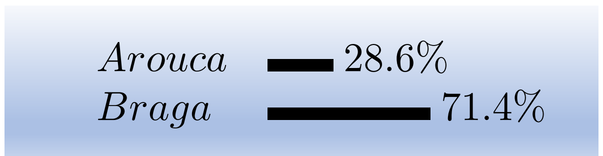


$$h_2\ (x, y) = \mathbf{do}\ \{ a \leftarrow x; b \leftarrow y; f\ (a, b) \}$$

Exemplo – liga de futebol

```
jogo :: (Equipa, Equipa) -> Dist Equipa

*Main> jogo ("Braga","Arouca")
 "Braga"   71.4%
"Arouca"   28.6%
```

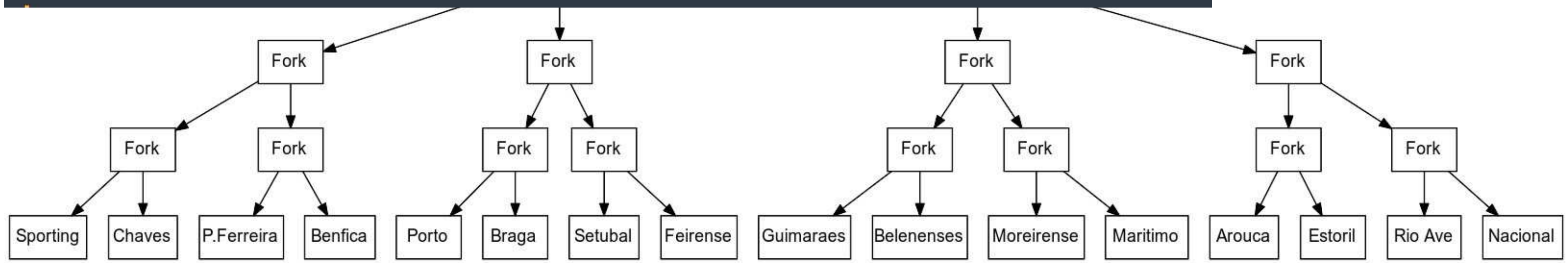




etc

Exemplo – liga de futebol

```
h2(d1,d2) = do { a <- d1; b <- d2; jogo(a,b) }

simular  = cataLTree (either return h2)
```




```
calendario = anaLTree lsplit equipas
```

Exemplo – liga de futebol


```
*Main> simular calendario
      "Porto"   24.0%
    "Benfica"   22.0%
   "Sporting"   19.8%
      "Braga"    6.0%
  "Guimaraes"    4.5%
 "Belenenses"    3.7%
   "Nacional"    3.7%
   "Maritimo"    3.5%
 "Moreirense"    2.3%
    "Rio Ave"    2.3%
    "Setubal"    2.1%
 "P.Ferreira"    1.8%
     "Arouca"    1.2%
     "Chaves"    1.2%
   "Feirense"    1.1%
    "Estoril"    0.8%
```

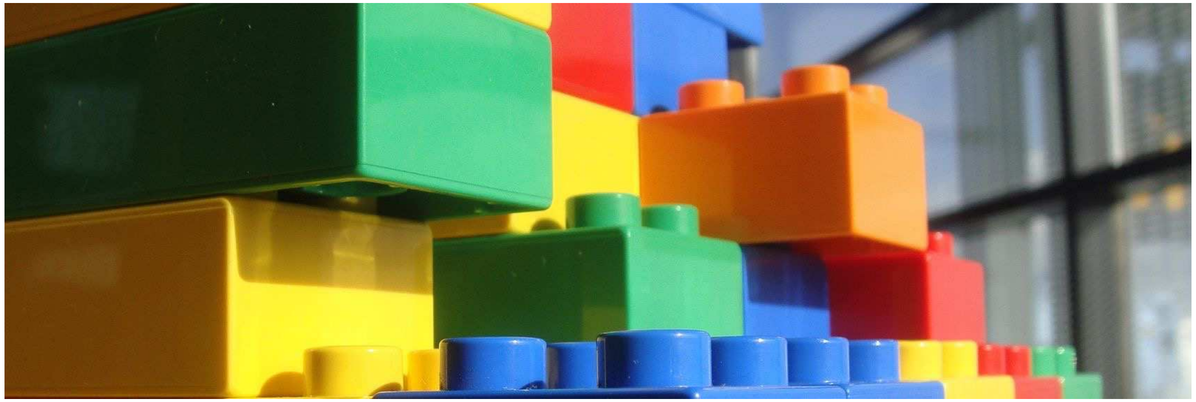
etc

Monad =
functor "de corridas"

$$X \xrightarrow{u} \mathbf{T}\,X \xleftarrow{\mu} \mathbf{T}\,(\mathbf{T}\,X)$$

Considerações finais

CÁLCULO DE PROGRAMAS

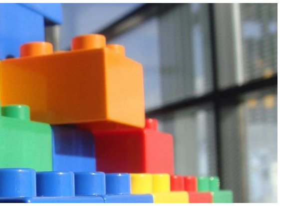


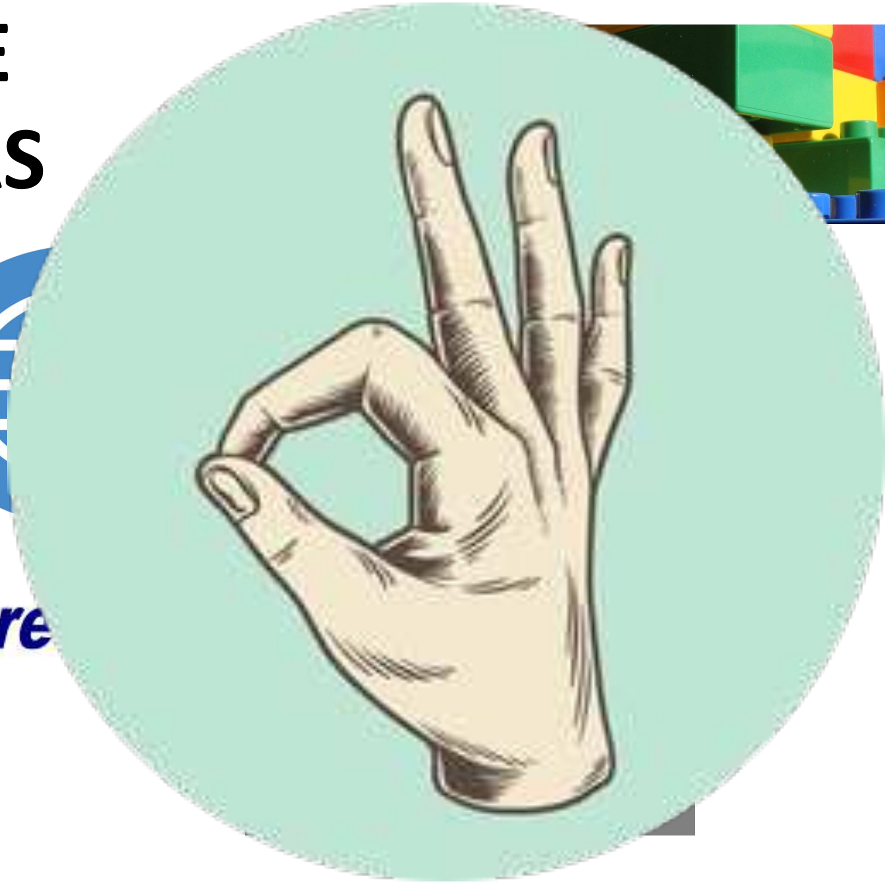
CÁLCULO DE PROGRAMAS

Software Reuse

CÁLCULO DE PROGRAMAS

CÁLCULO DE PROGRAMAS





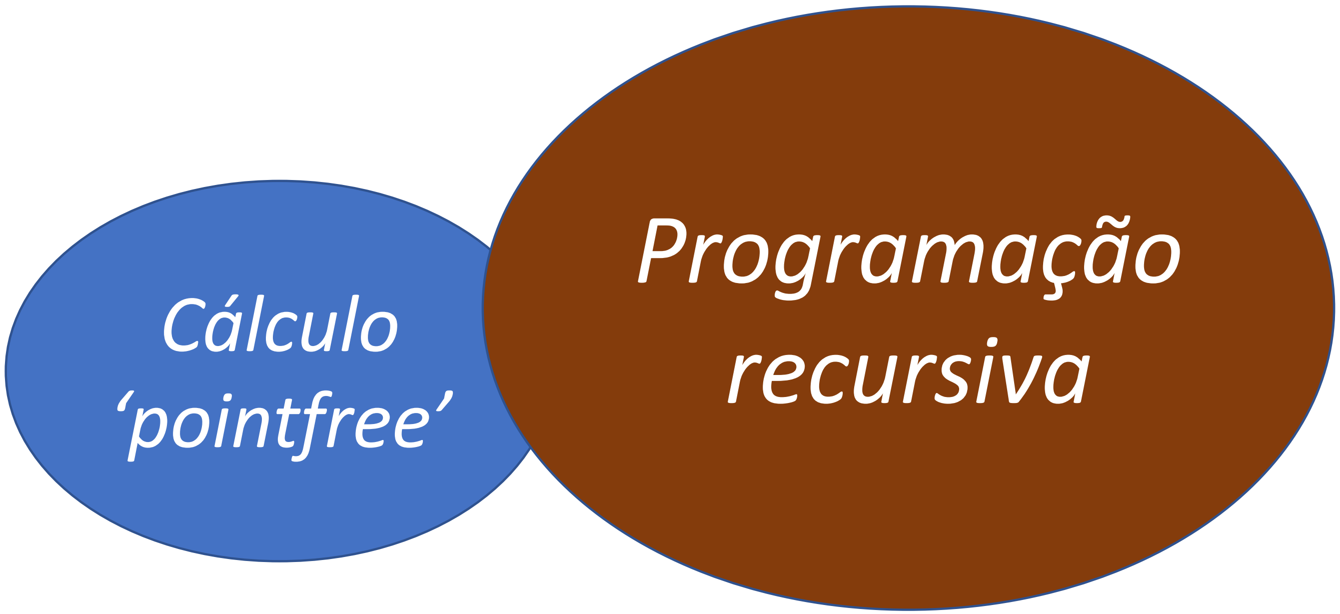
Software
Reuse

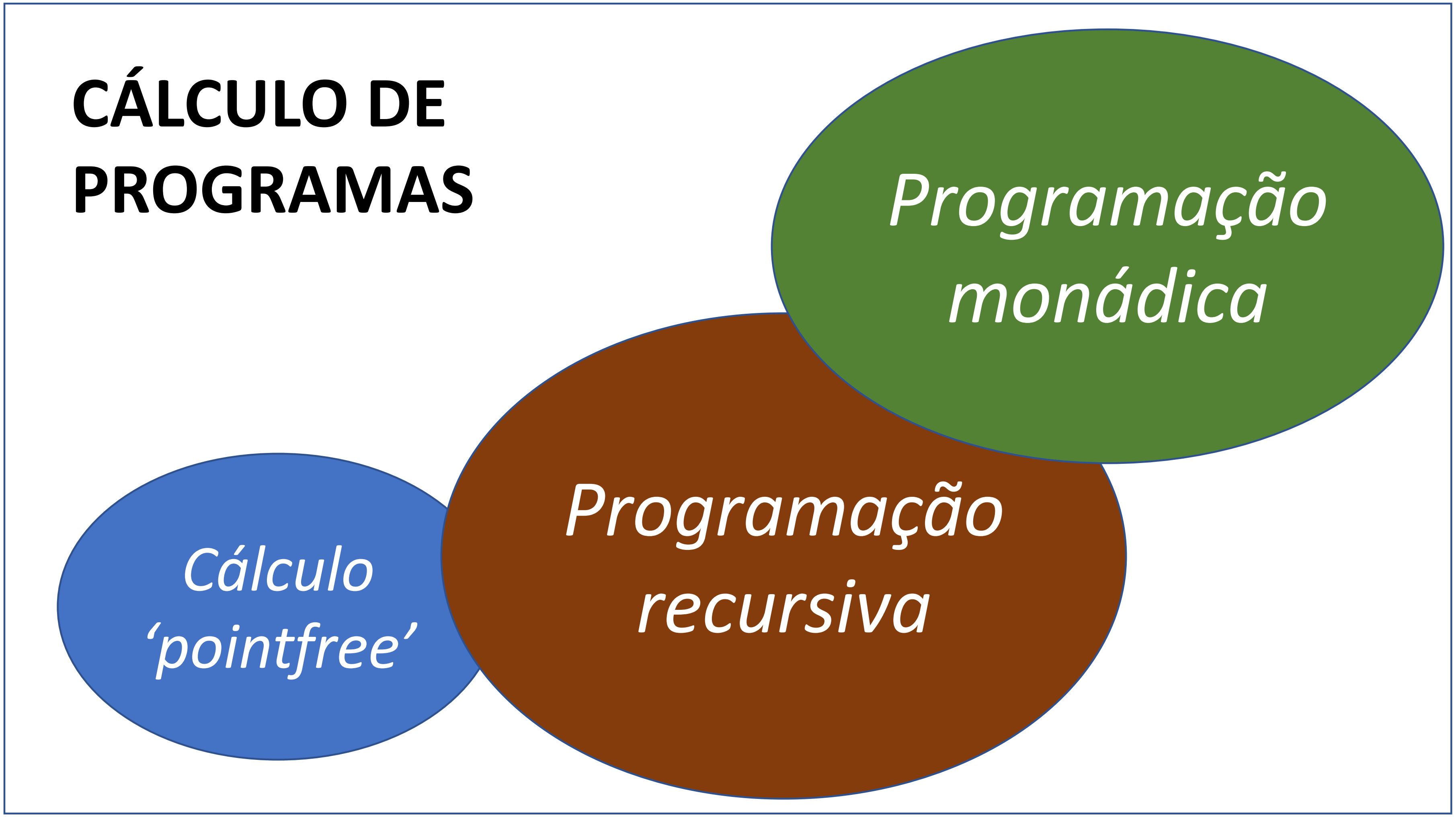
COPY
PASTE

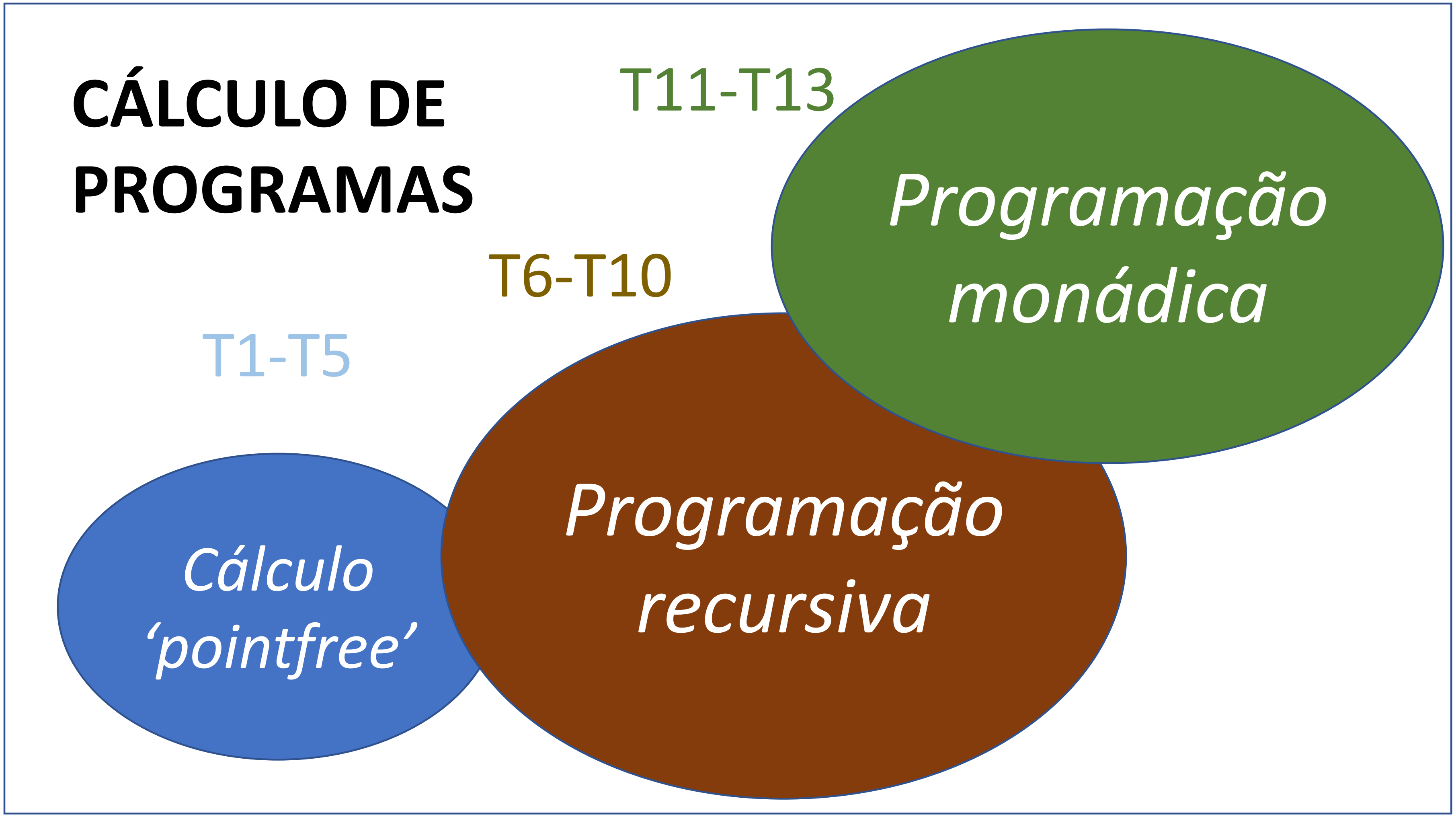
CÁLCULO DE PROGRAMAS

Cálculo 'pointfree'

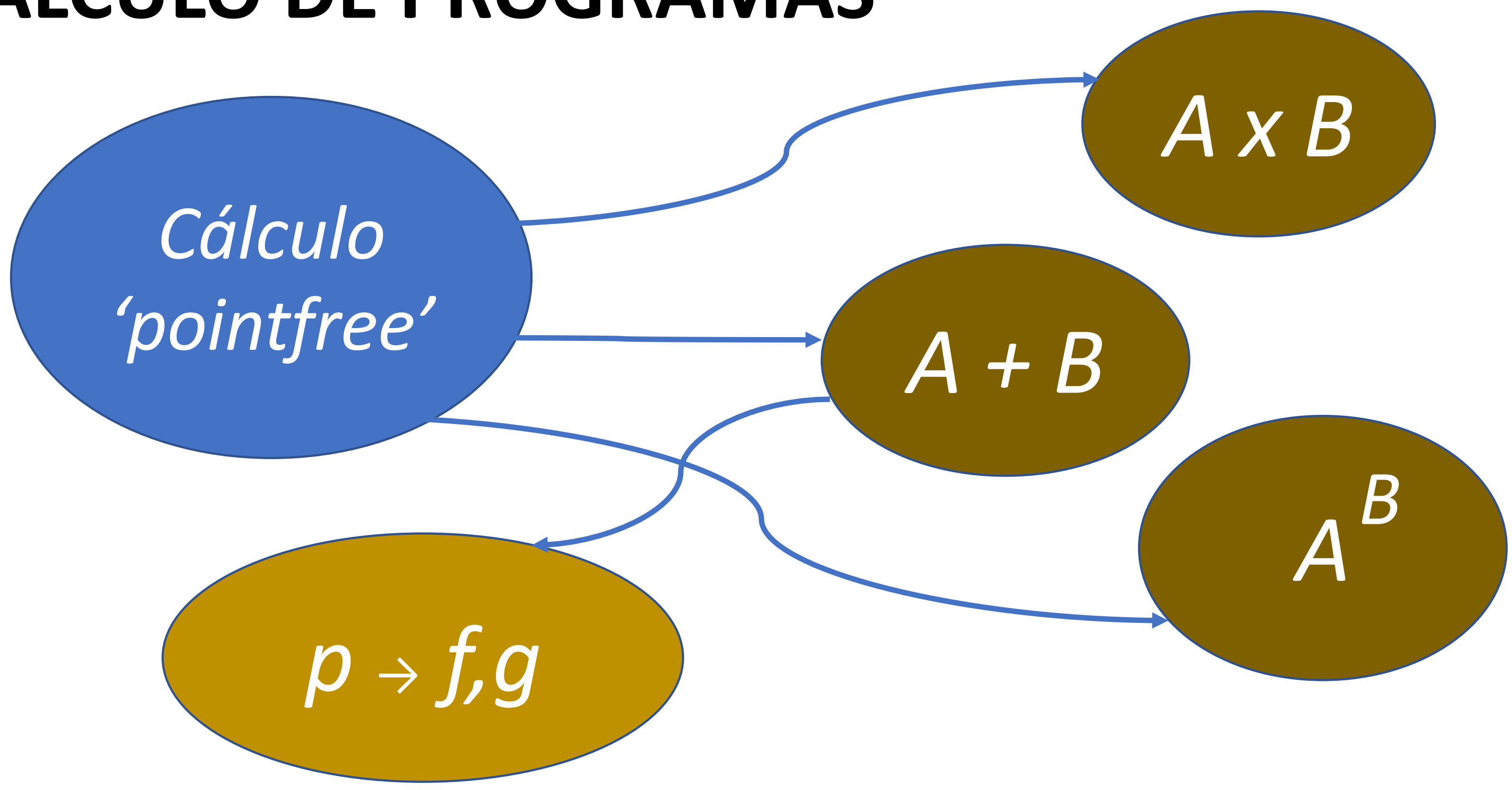
CÁLCULO DE PROGRAMAS

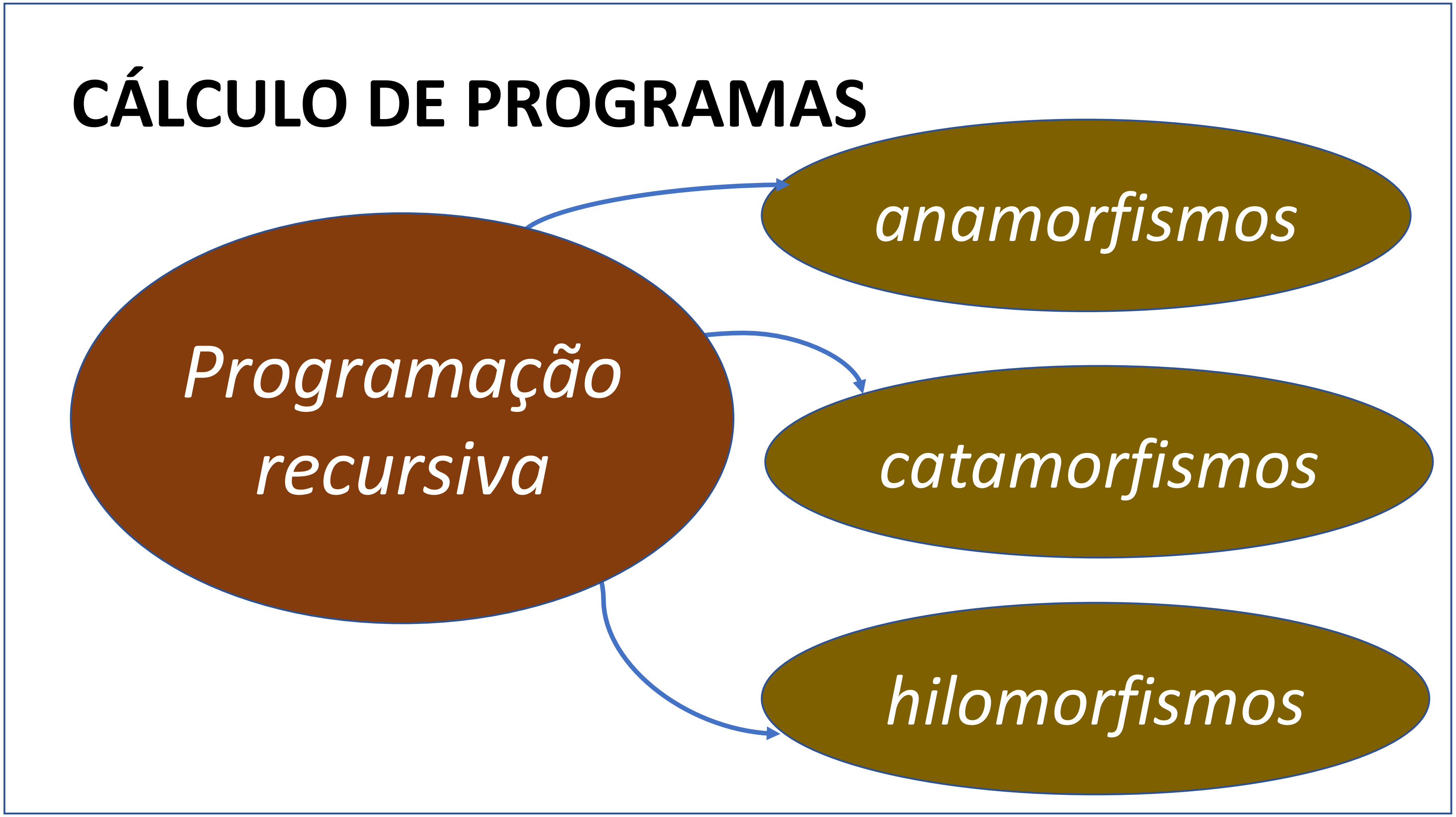



CÁLCULO DE PROGRAMAS
Programação monádica
Programação recursiva
Cálculo ‘pointfree’

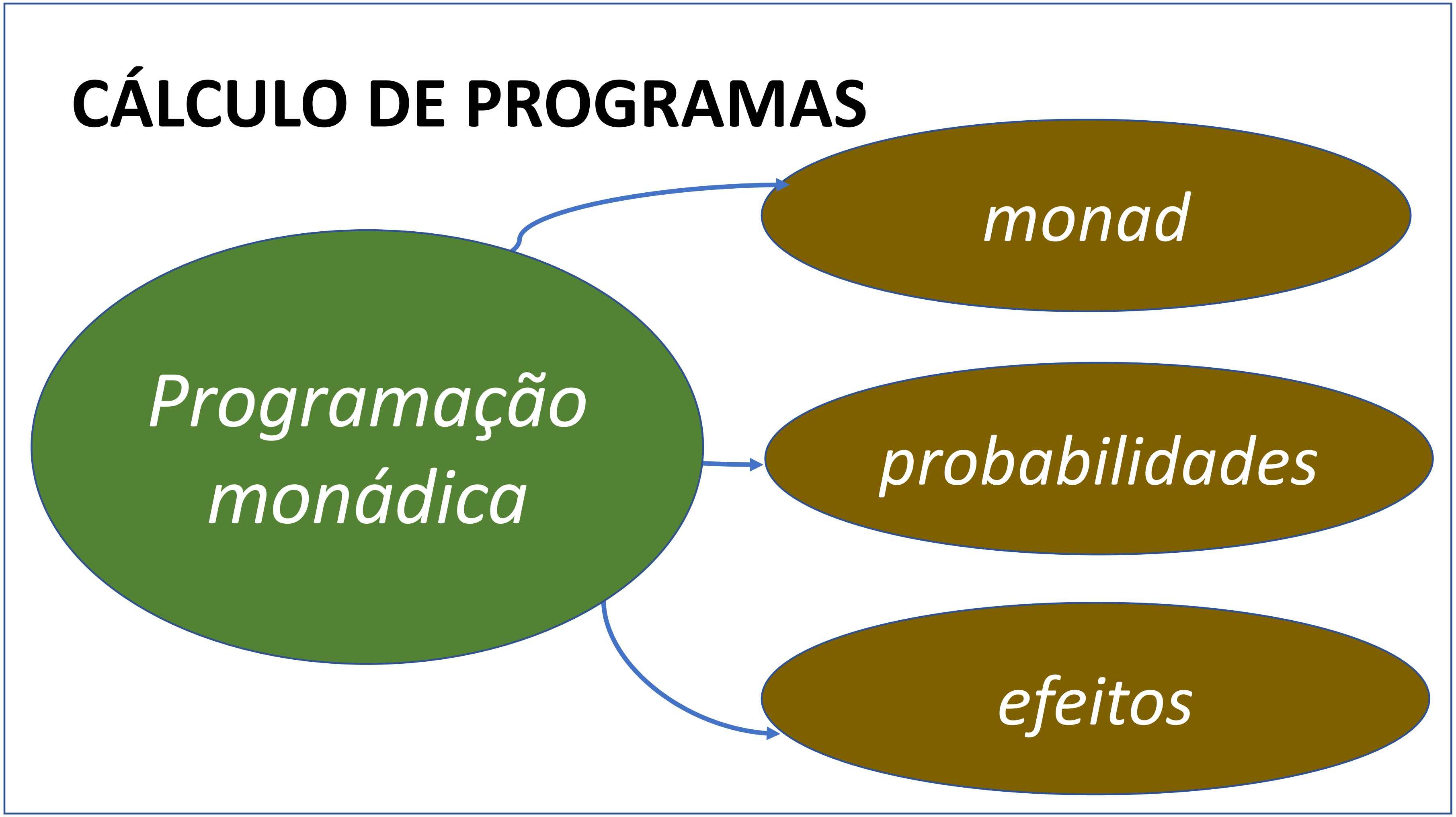

CÁLCULO DE PROGRAMAS
T11-T13
T6-T10
T1-T5
Programação monádica
Programação recursiva
Cálculo 'pointfree'

CÁLCULO DE PROGRAMAS




CÁLCULO DE PROGRAMAS
Programação recursiva
anamorfismos
catamorfismos
hilomorfismos

CÁLCULO DE PROGRAMAS


CÁLCULO DE PROGRAMAS
Programação monádica
monad
probabilidades
efeitos