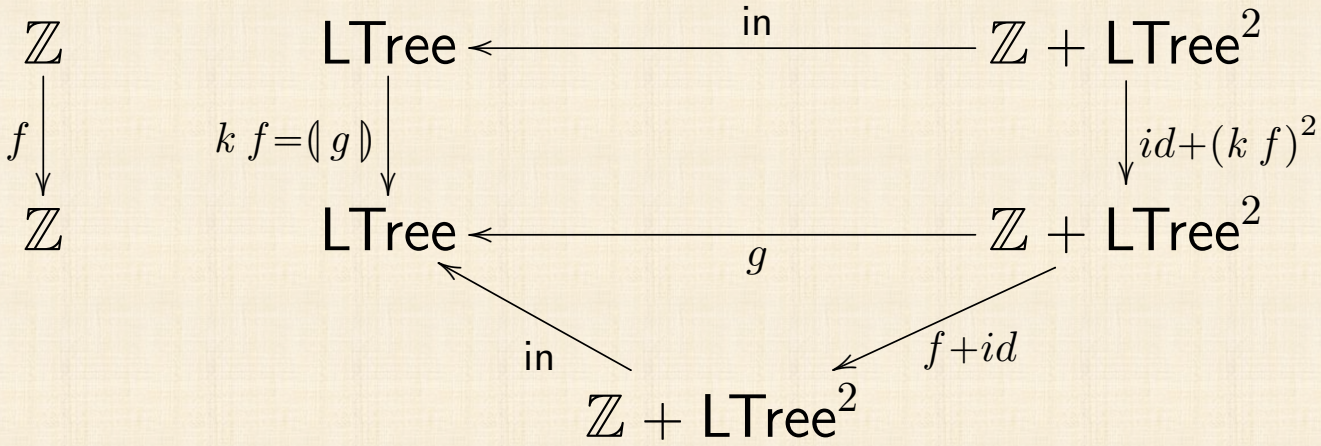
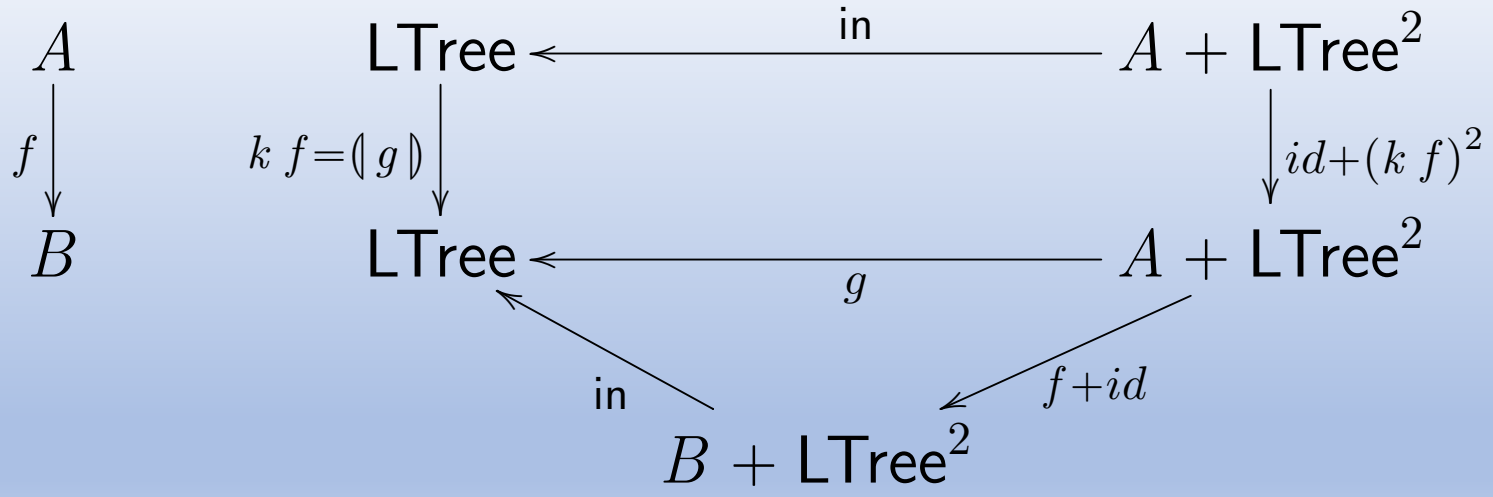
Cálculo de Programas
T09 (cntd)

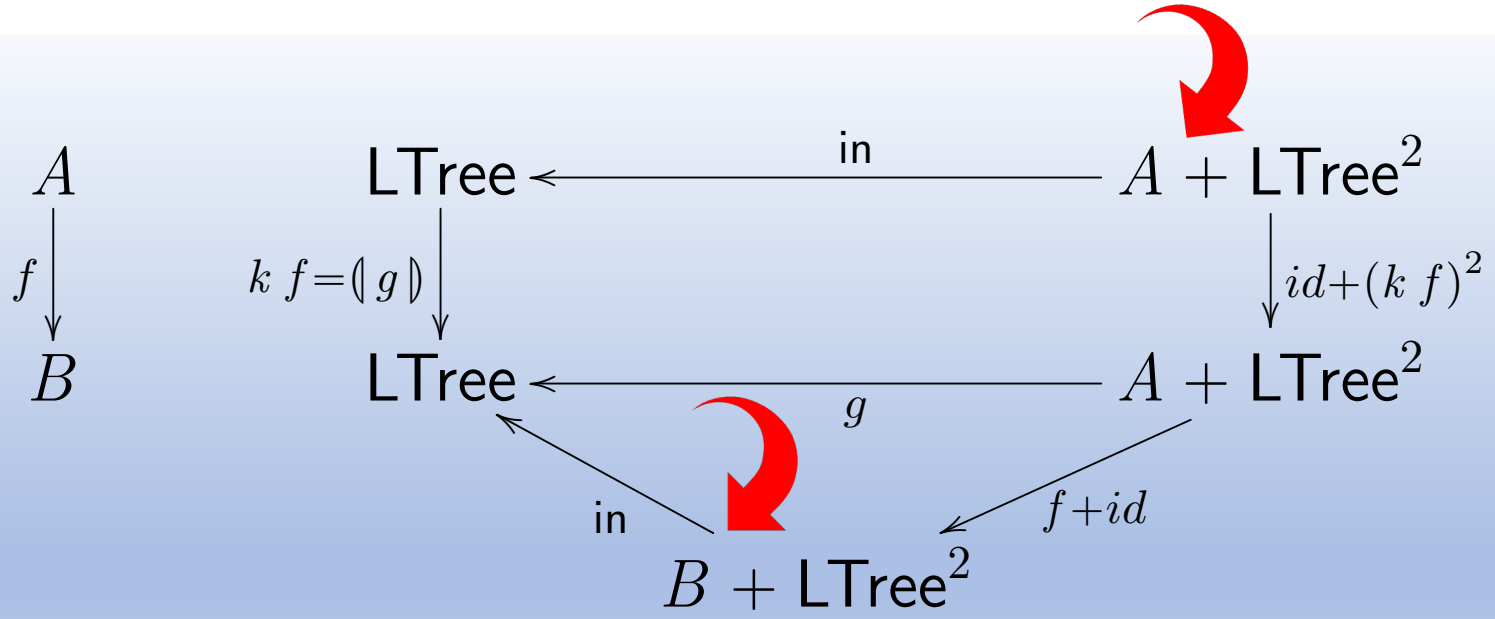
$$k\ f \cdot \mathsf{in} = \mathsf{in} \cdot (f + id) \cdot (id + k\ f \times k\ f)$$



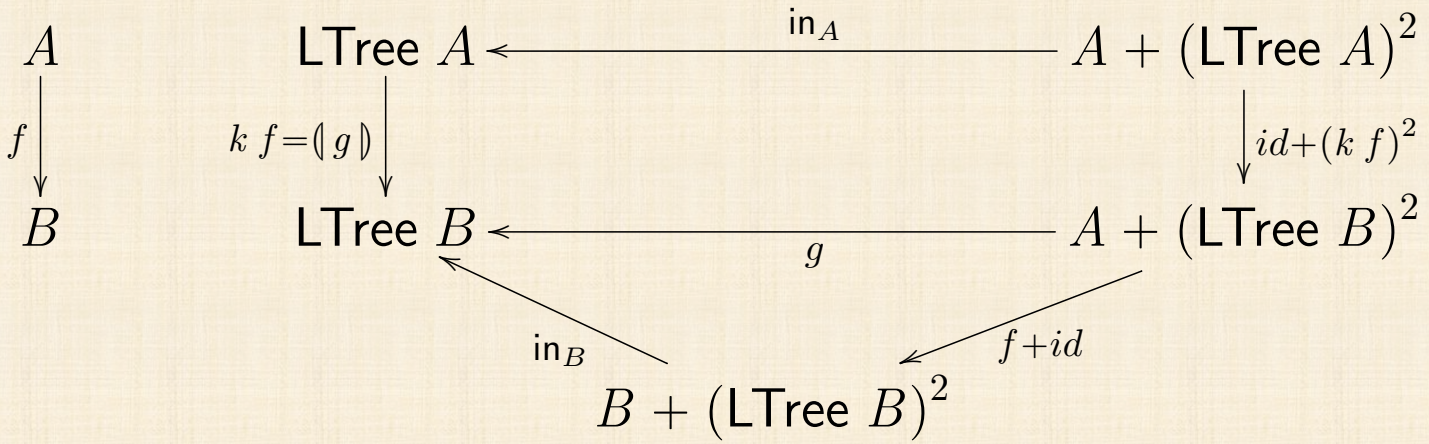
$$k\ f \cdot \mathsf{in} = \mathsf{in} \cdot (f + id) \cdot (id + k\ f \times k\ f)$$



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



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



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

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

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

Absorption:

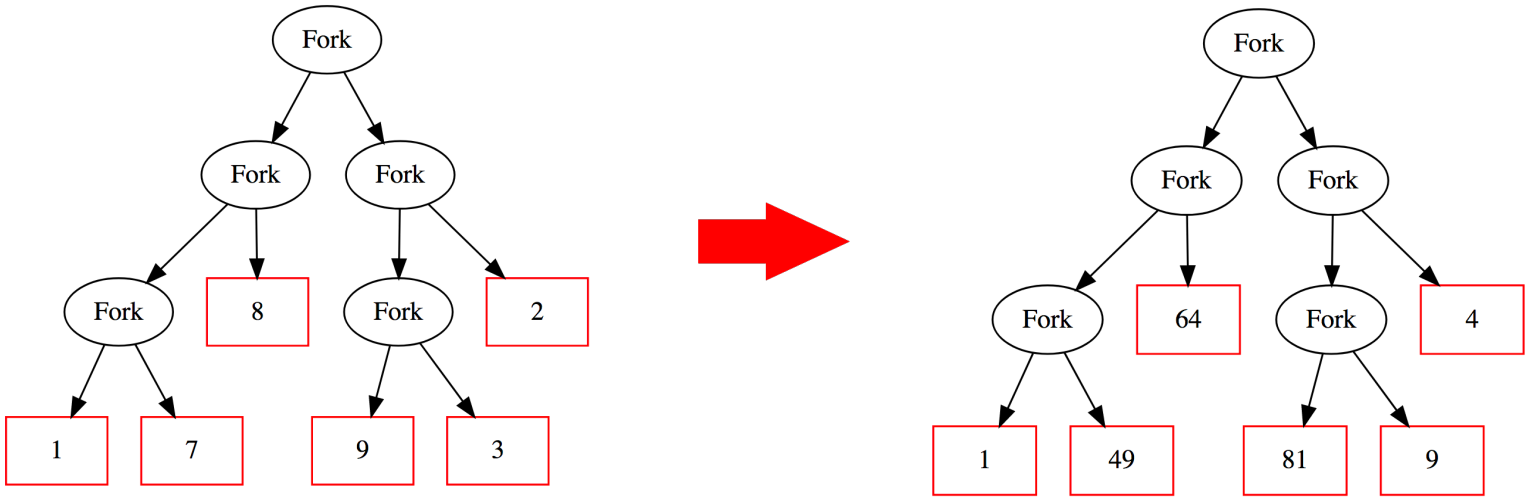
$$(\!|\ g\ |\!) \cdot k\ f = (\!|\ g \cdot (f + id)\ |\!)$$

$$
\begin{aligned}
& (\!|\, g \,|\!) \cdot k\, f = (\!|\, g \cdot (f + id) \,|\!) \\
\equiv & \qquad \{\ k\, f = (\!|\, \mathsf{in} \cdot (f + id) \,|\!)\ \} \\
& (\!|\, g \,|\!) \cdot (\!|\, \mathsf{in} \cdot (f + id) \,|\!) = (\!|\, g \cdot (f + id) \,|\!) \\
\Leftarrow & \qquad \{\ \text{cata-fusion}\ \} \\
& (\!|\, g \,|\!) \cdot \mathsf{in} \cdot (f + id) = g \cdot (f + id) \cdot (id + (\!|\, g \,|\!)^2) \\
\equiv & \qquad \{\ \text{cata-cancellation}\ \} \\
& g \cdot (id + (\!|\, g \,|\!)^2) \cdot (f + id) = g \cdot (f + id) \cdot (id + (\!|\, g \,|\!)^2) \\
\equiv & \qquad \{\ \text{functor-}+\ \text{twice; natural-}id\ \text{four times}\ \} \\
& g \cdot (f + (\!|\, g \,|\!)^2) = g \cdot (f + (\!|\, g \,|\!)^2) \\
\equiv & \qquad \{\ \text{trivial}\ \} \\
& true
\end{aligned}
$$

$$
\begin{aligned}
& k\ (f \cdot g) \\
= & \quad \{\ k\ f = (\!|\ \mathsf{in} \cdot (f + id)\ |\!)\ \} \\
& (\!|\ \mathsf{in} \cdot (f \cdot g + id)\ |\!) \\
= & \quad \{\ +\text{-functor etc}\ \} \\
& (\!|\ \mathsf{in} \cdot (f + id) \cdot (g + id)\ |\!) \\
= & \quad \{\ \text{absorption (previous slides)}\ \} \\
& (\!|\ \mathsf{in} \cdot (f + id)\ |\!) \cdot (\!|\ \mathsf{in} \cdot (g + id)\ |\!) \\
= & \quad \{\ k\ f = (\!|\ \mathsf{in} \cdot (f + id)\ |\!)\ \text{twice}\ \} \\
& k\ f \cdot k\ g
\end{aligned}
$$

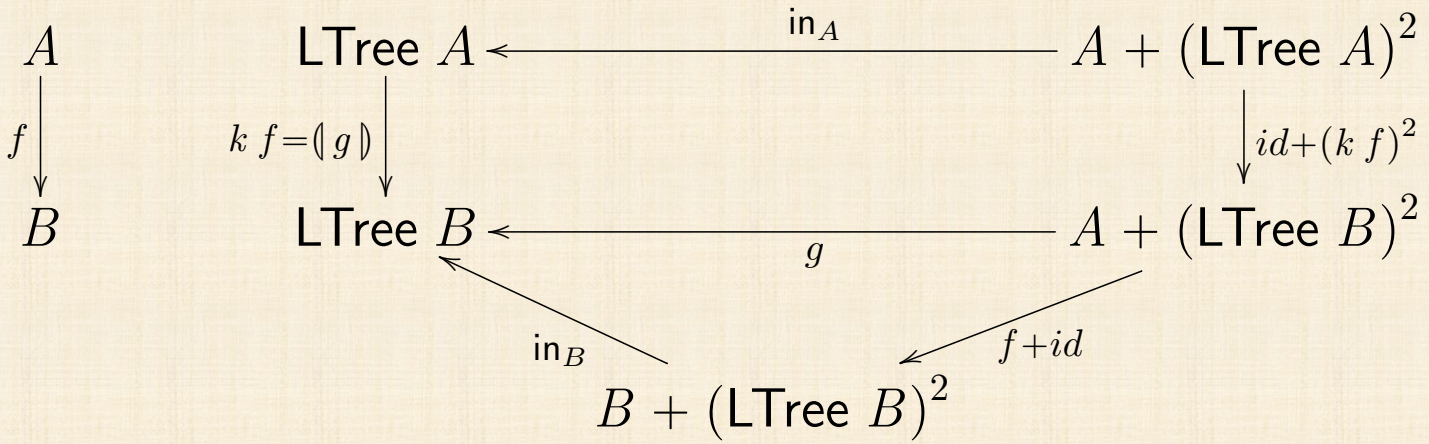
$$k\ id = (\!|\ \mathsf{in}\ |\!) = id$$

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

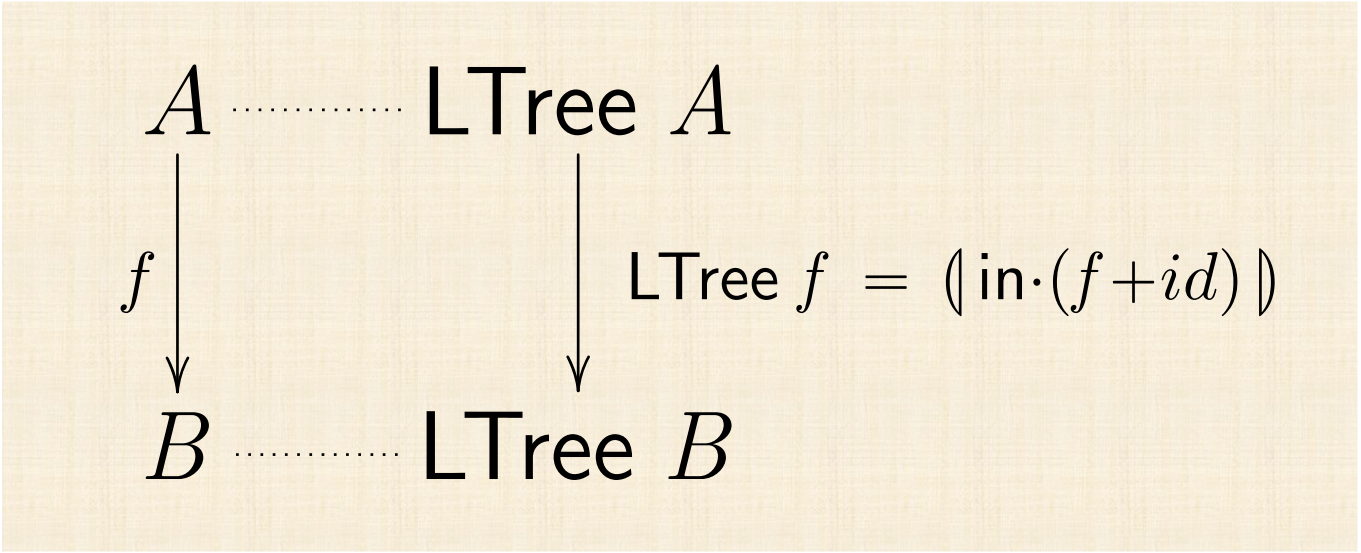



```
data LTree = Leaf Int | Fork (LTree, LTree)

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

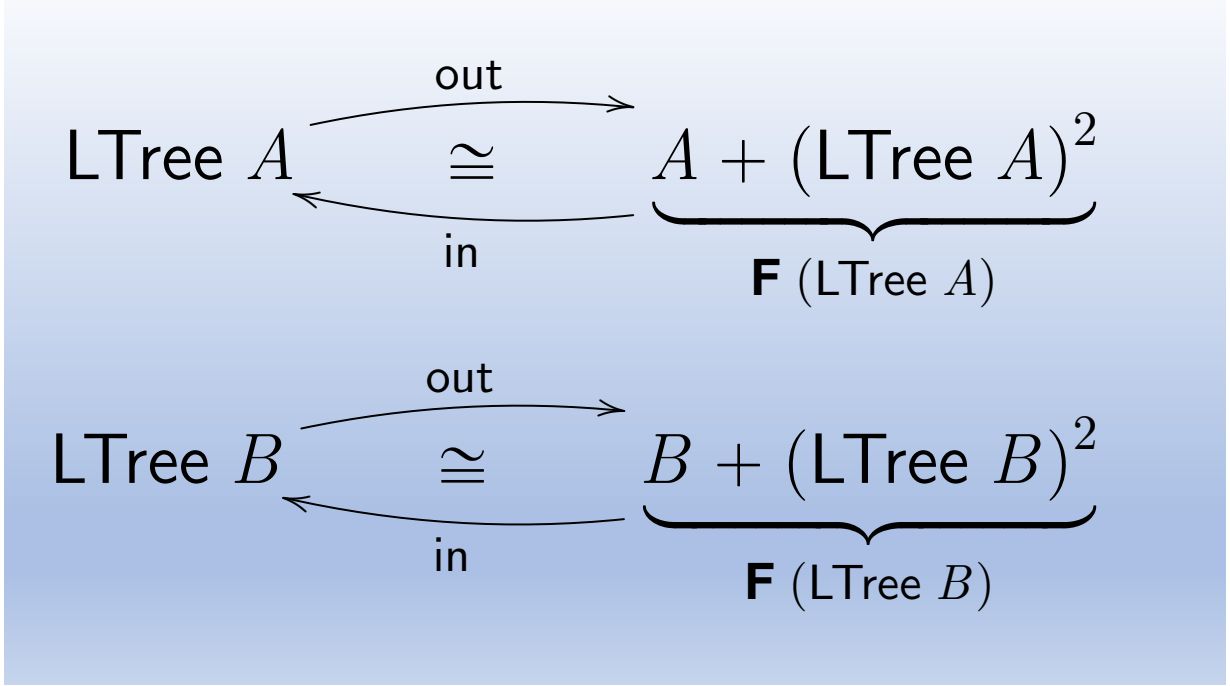
$$k\ f = (\!|\ \mathsf{in} \cdot (f + id)\ |\!)$$



LTREE TYPE FUNCTOR



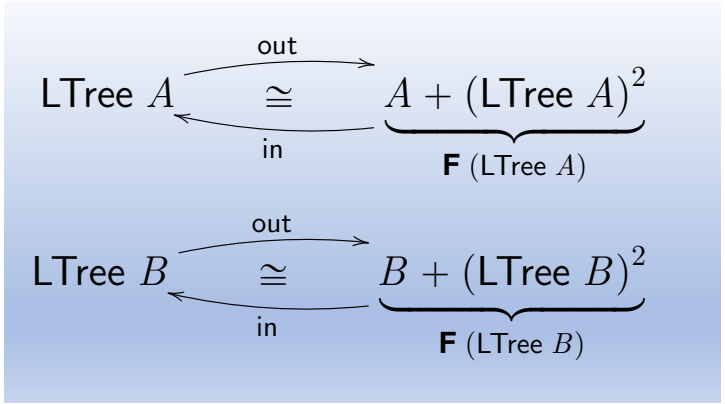
LTREE TYPE FUNCTOR



LTREE TYPE FUNCTOR

$$\mathbf{F}\ X = A + X^2 \quad ?$$

$$\mathbf{F}\ X = B + X^2 \quad ?$$

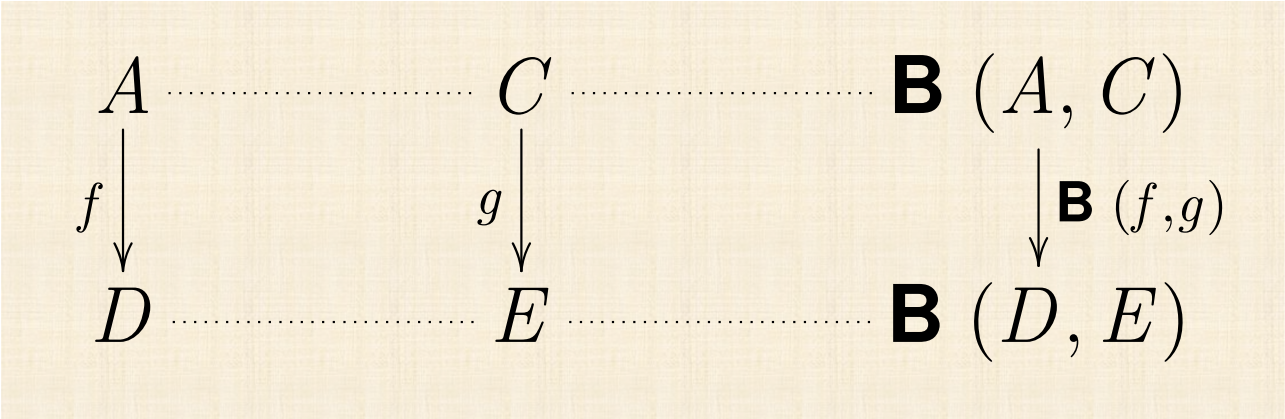


LTREE base bifunctor B

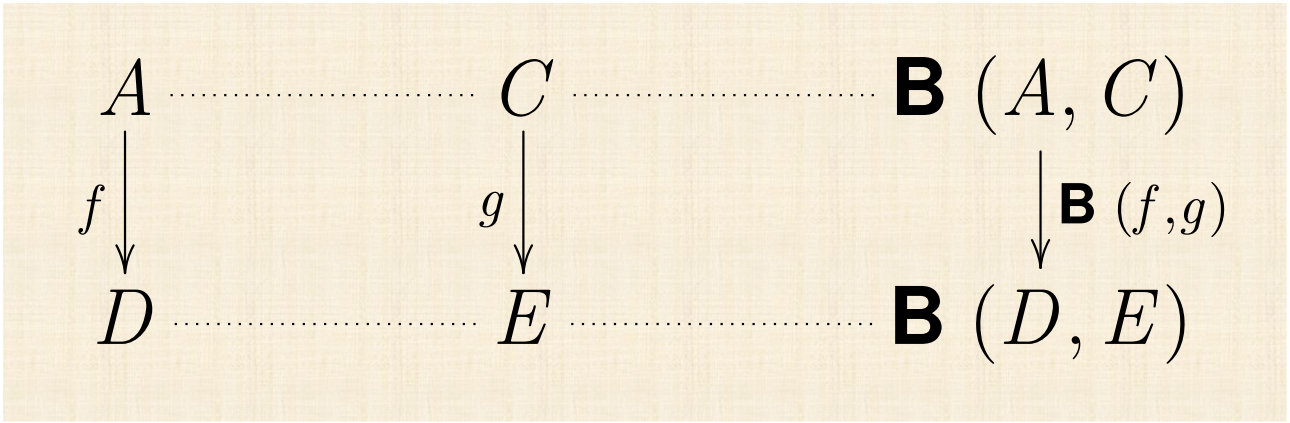
$$\mathsf{LTree}\ X \underset{\text{in}}{\overset{\text{out}}{\cong}} \underbrace{X + (\mathsf{LTree}\ X)^2}_{\mathbf{B}\ (X, \mathsf{LTree}\ X)}$$

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

BIFUNCTORS



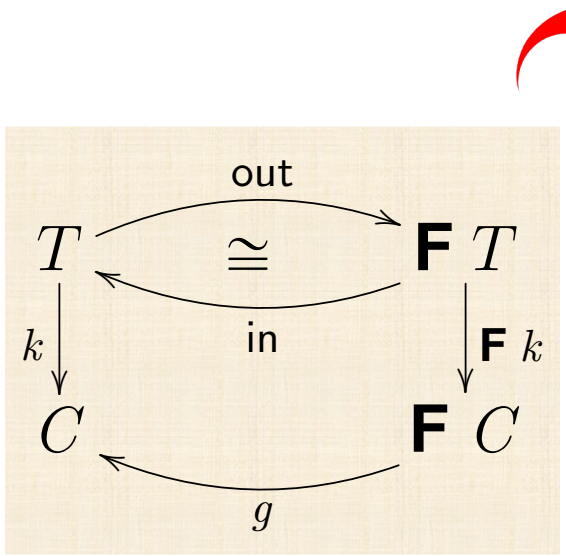
BIFUNCTORS (Laws)

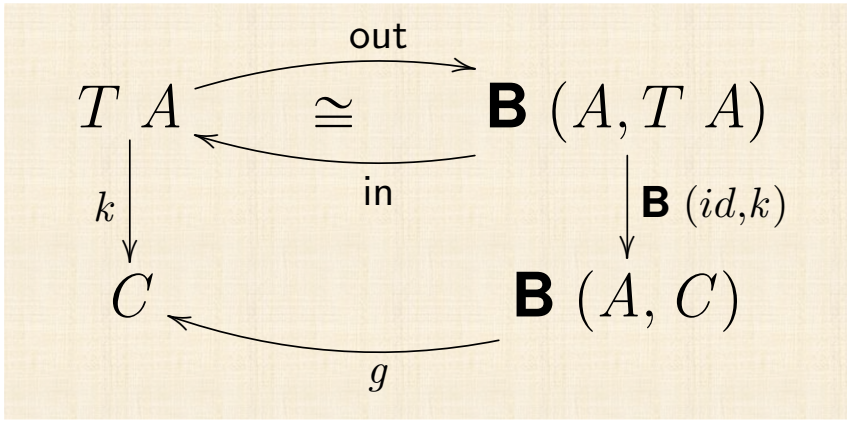


$$\mathbf{B}\,(id, id) = id$$

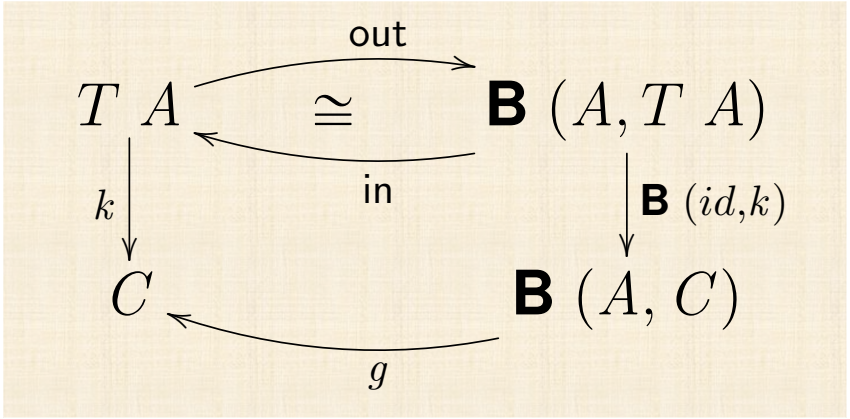
$$\mathbf{B}\,(h \cdot f, k \cdot g) = \mathbf{B}\,(h, k) \cdot \mathbf{B}\,(f, g)$$

CATAMORPHISMS (generalization)





CATAMORPHISMS (in general)



Universal property

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

Type functor:

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

Abbreviation:

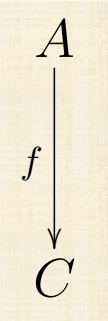
$$\mathbf{F}\ k = \mathbf{B}\ (id, k)$$

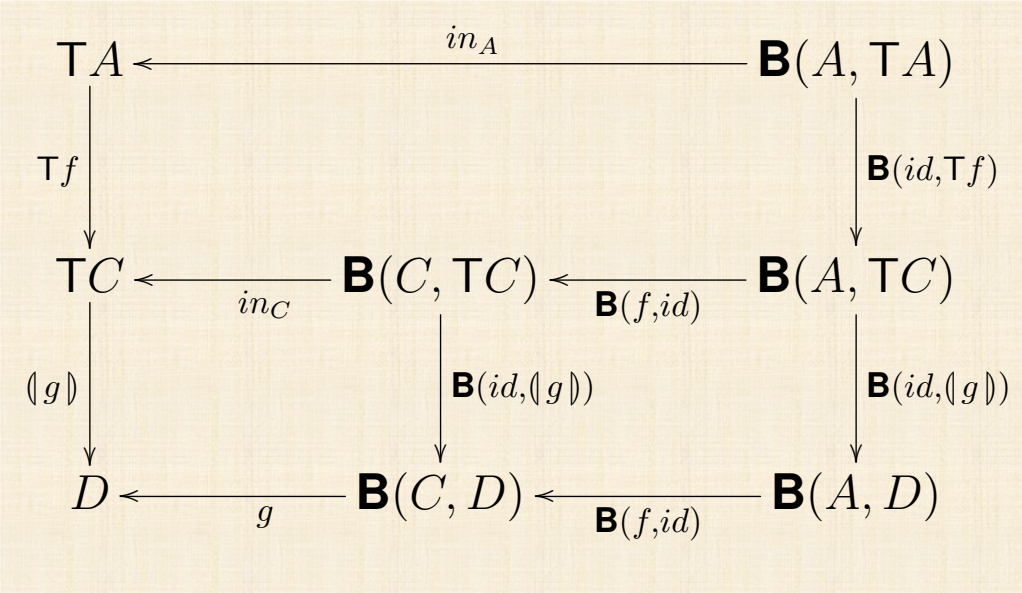
CATAMORPHISMS (Laws)

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

Cata-absorption

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





(a) Trees whose data of type A are stored in their nodes:

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

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

(b) Trees with data in their leafs :

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

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

(c) Full trees — data in both leaves and nodes:

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

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

(d) Expression trees:

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

Haskell: $\mathbf{data}\ \mathsf{Expr}\ v\ o = Var\ v \mid Op\ (o, [\mathsf{Expr}\ v\ o])$

(a) Trees whose data of type A are stored in their nodes:

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

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

(b) Trees with data in their leafs :

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

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

(c) Full trees — data in both leaves and nodes:

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

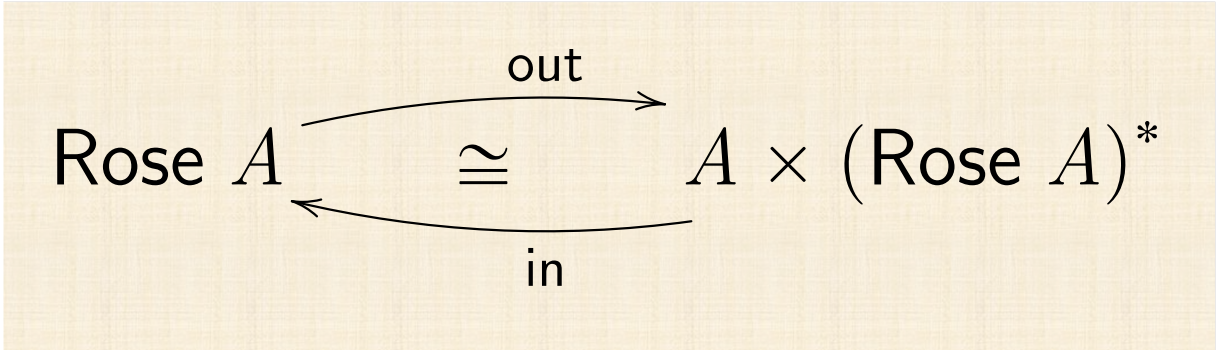
Haskell: $\mathbf{data}\ \mathsf{FTree}\ b\ a = Unit\ b \mid Comp\ (a, (\mathsf{FTree}\ b\ a, \mathsf{FTree}\ b\ a))$

(d) Expression trees:

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

Haskell: $\mathbf{data}\ \mathsf{Expr}\ v\ o = Var\ v \mid Op\ (o, [\mathsf{Expr}\ v\ o])$

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

inRose = uncurry Rose

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



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

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

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

$$A \times Y^*$$

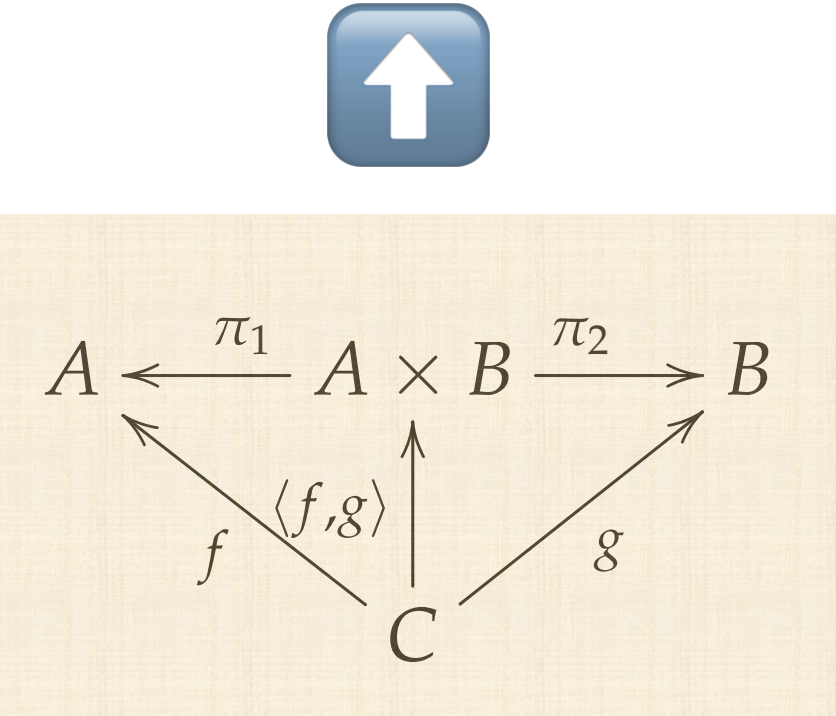
$$X \times Y^*$$

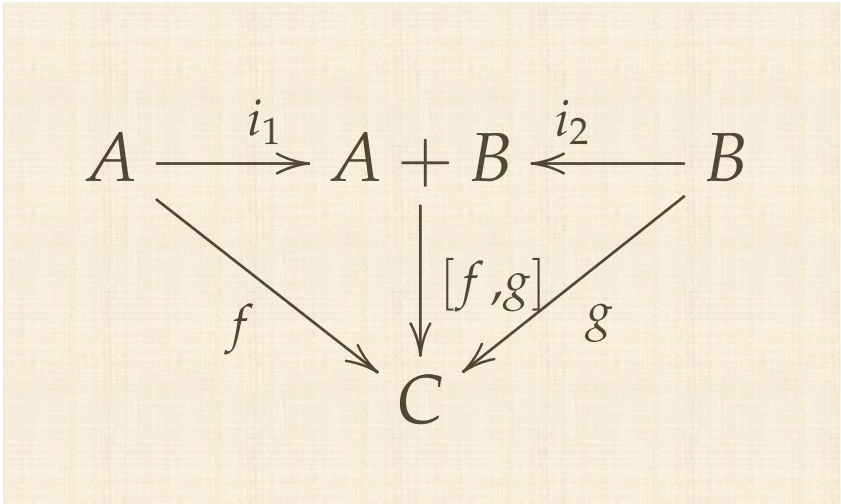
```haskell
data Rose a = Rose a [Rose a] deriving Show

inRose = uncurry Rose

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

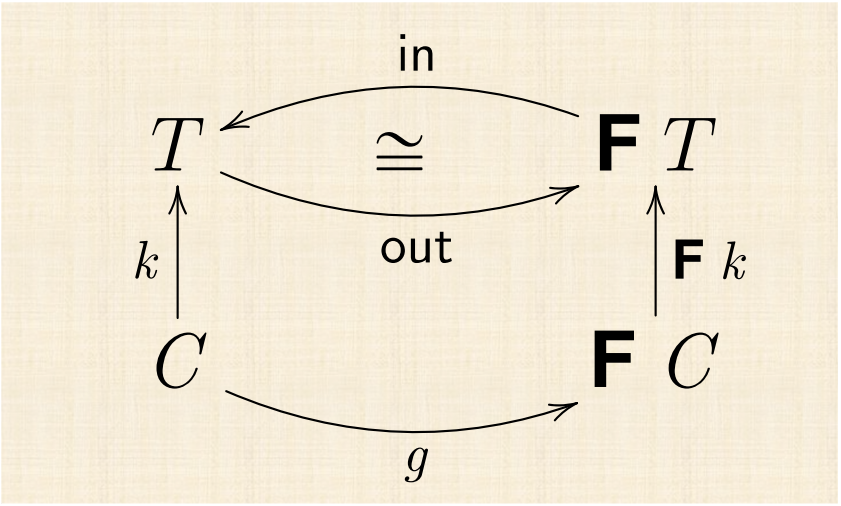
$$\mathbf{B}\,(X, Y) = X \times Y^*$$
$$\mathbf{B}\,(f, g) = f \times g^*$$

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

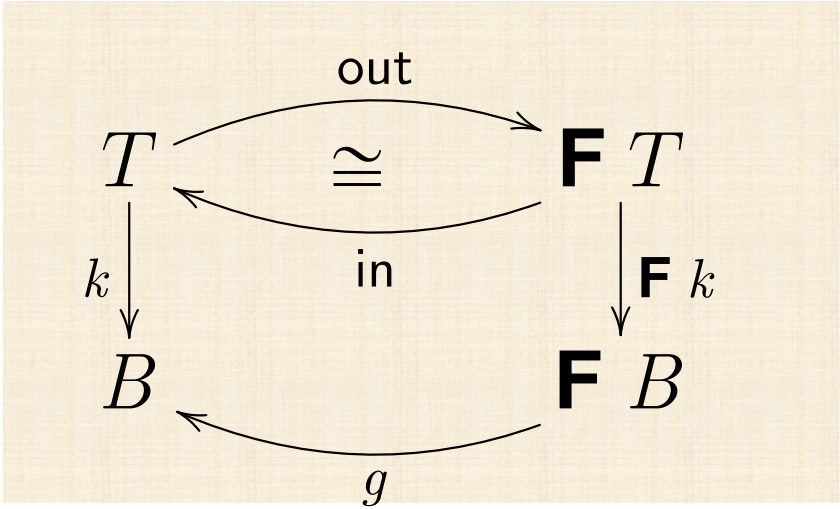
Anamorphisms


A
π1
A × B
π2
B
⟨f,g⟩
f
g
C


A
i1
A + B
i2
B
[f,g]
f
g
C

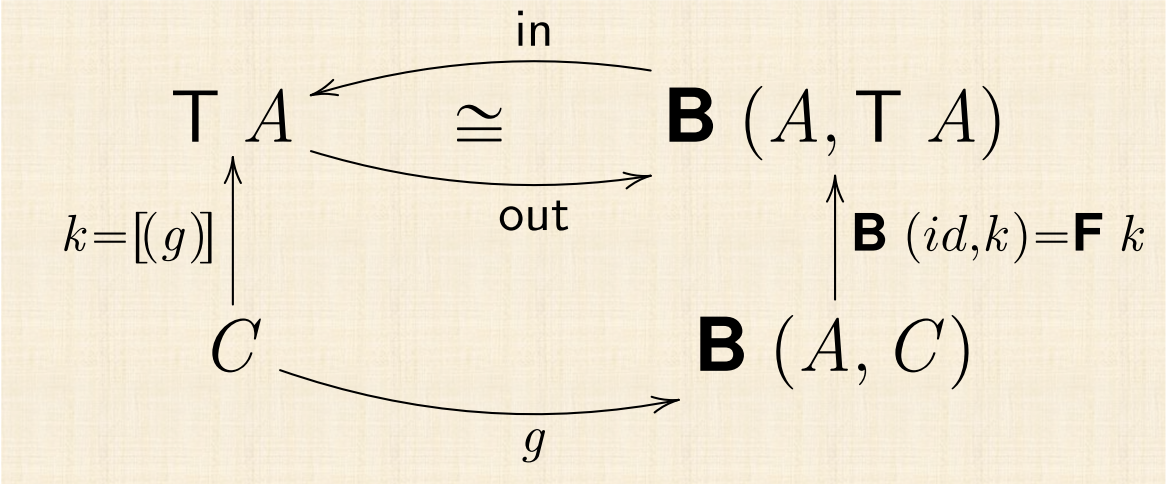


$\alpha\nu\alpha$ (ana)



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

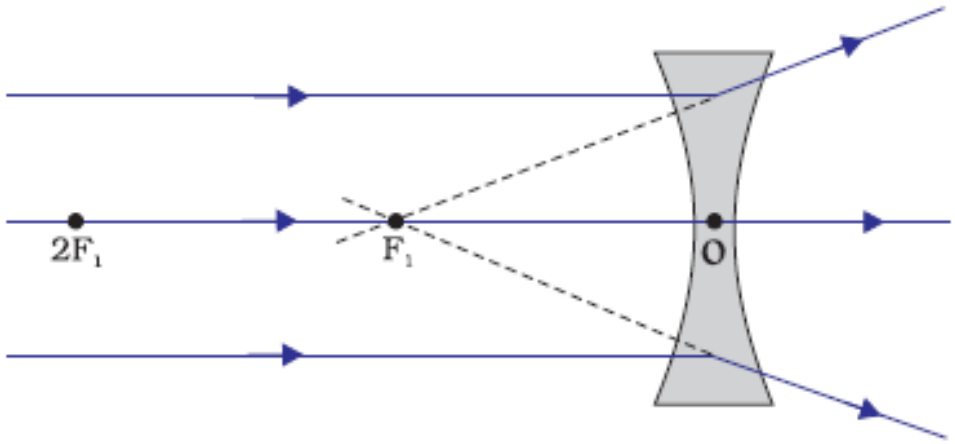
ANAMORPHISMS



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

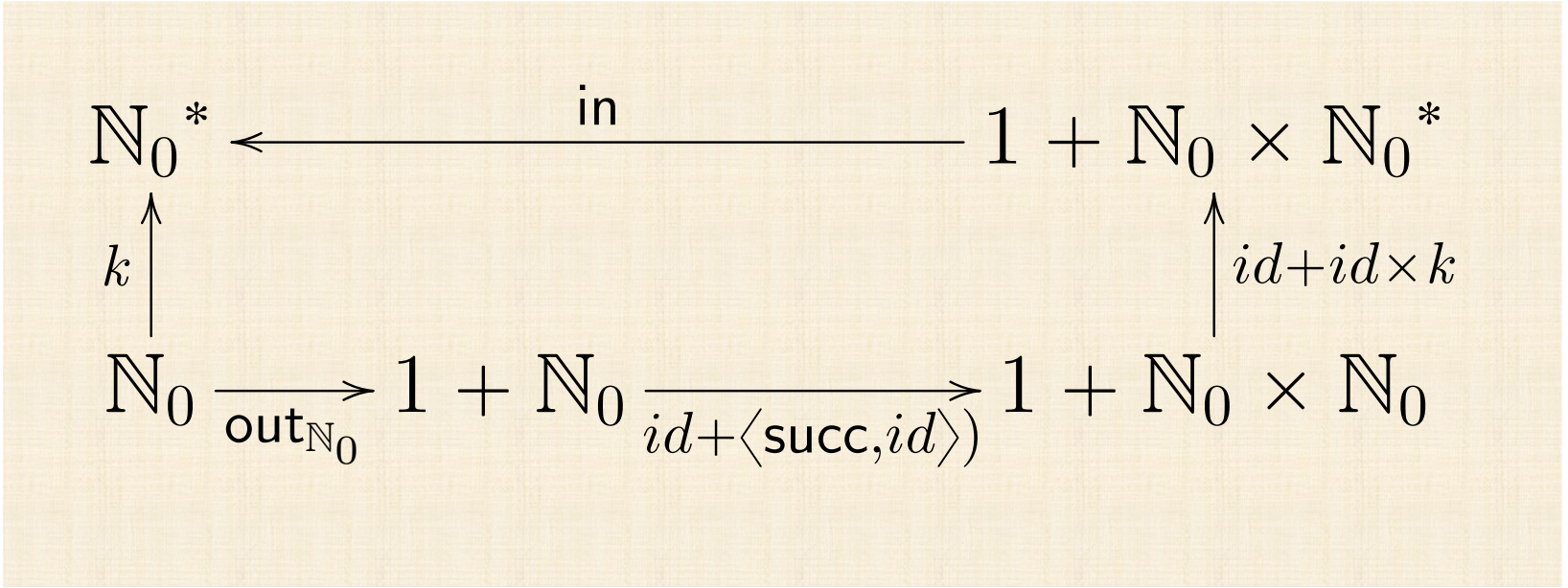
ANAMORPHISM LAWS

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

[(–)]

Example


$\mathbb{N}_0{}^*$
in
$1 + \mathbb{N}_0 \times \mathbb{N}_0{}^*$
k
$id+id\times k$
$\mathbb{N}_0$
$\mathsf{out}_{\mathbb{N}_0}$
$1 + \mathbb{N}_0$
$id+\langle \mathsf{succ}, id \rangle$
$1 + \mathbb{N}_0 \times \mathbb{N}_0$

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

$\equiv$ $\quad$ $\{$ definitions of in and $\mathsf{in}_{\mathbb{N}_0}$; fusion and absorption-$+$ $\}$

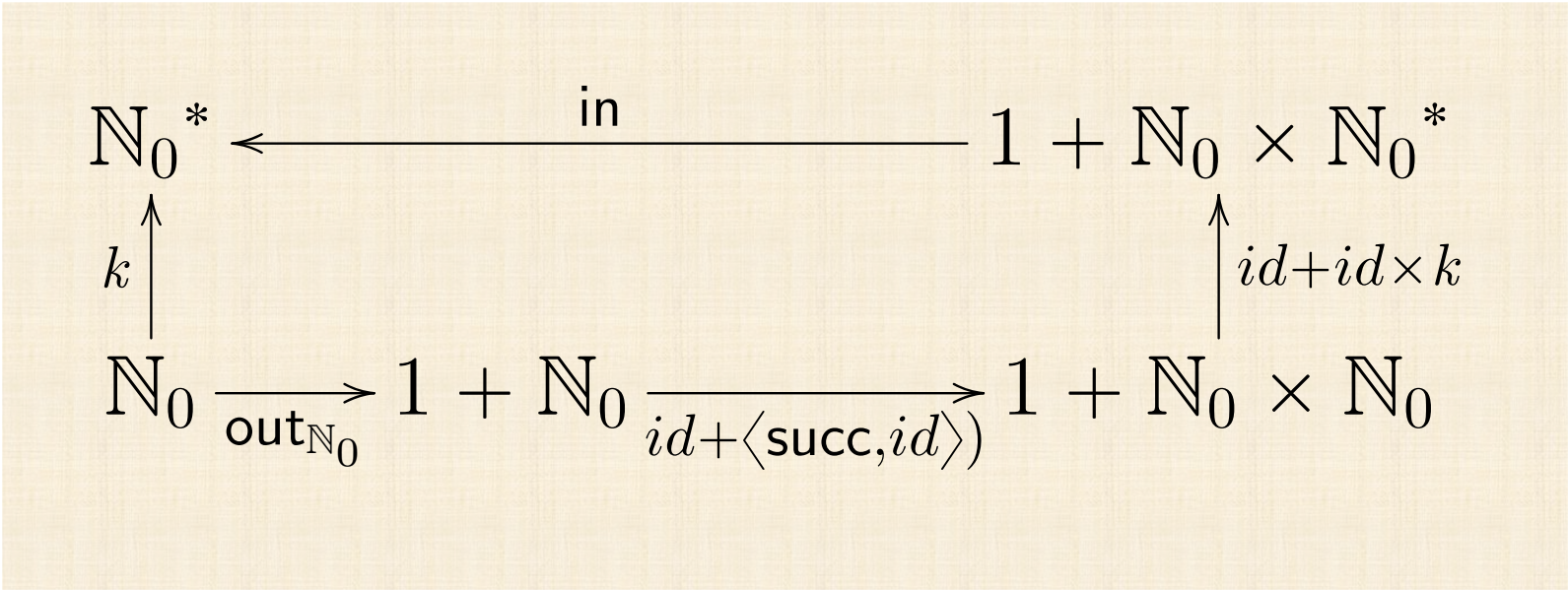
$$[k \cdot \underline{0}, k \cdot \mathsf{succ}] = [\mathsf{nil}, \mathsf{cons} \cdot \langle \mathsf{succ}, k \rangle]$$

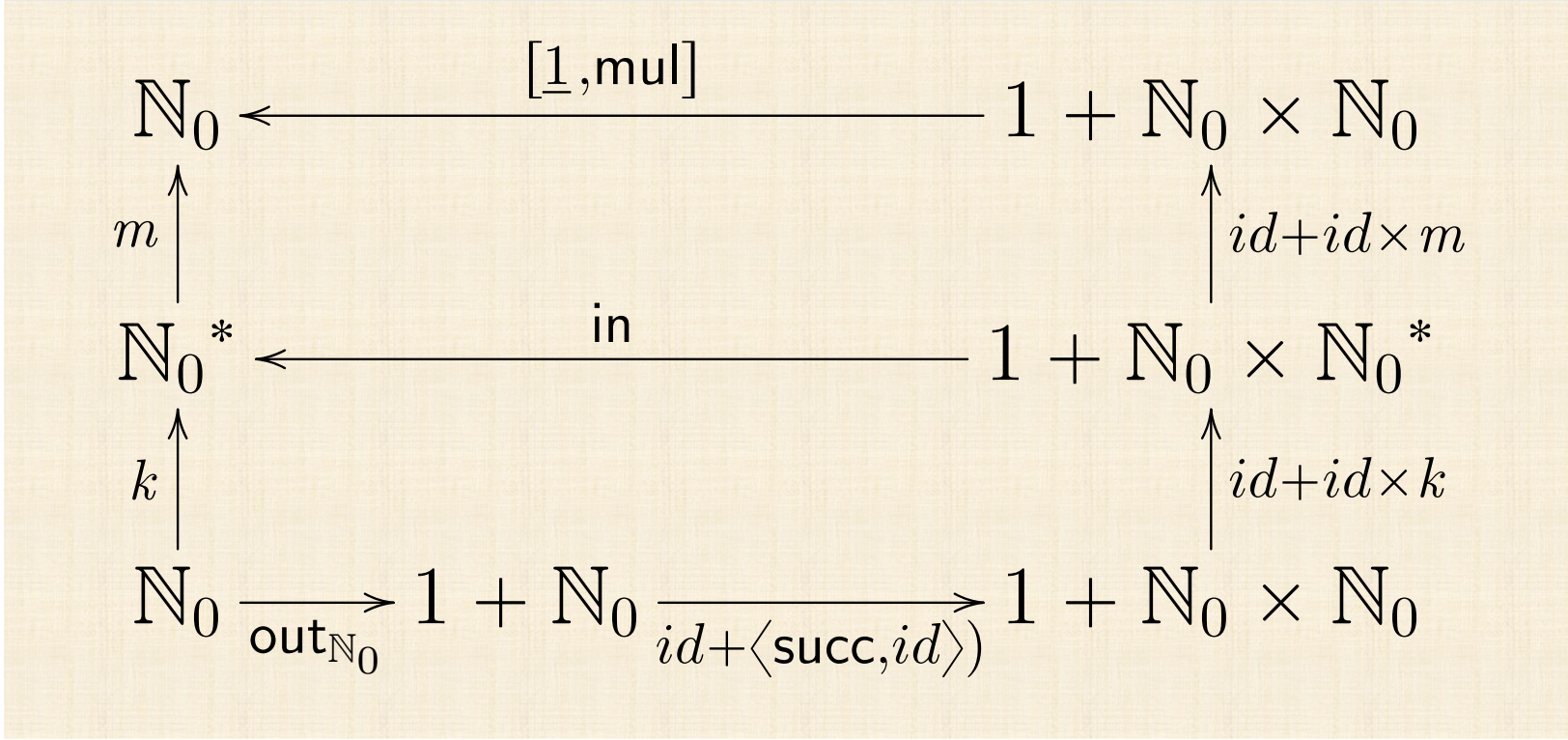
$\equiv$ $\quad$ $\{$ eq-$+$ $\}$

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

$\equiv$ $\quad$ $\{$ going pointwise $\}$

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


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


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

$$
\begin{aligned}
& f = m \cdot k \\
\equiv & \quad \{\ m = (\!|\, [\underline{1}, \mathsf{mul}]\, |\!) \text{ e } k = [\!(\, (id + \langle \mathsf{succ}, id \rangle) \cdot \mathsf{out}_{\mathbb{N}_0}\,)\!]\ \} \\
& f = (\!|\, [\underline{1}, \mathsf{mul}]\, |\!) \cdot [\!(\, (id + \langle \mathsf{succ}, id \rangle) \cdot \mathsf{out}_{\mathbb{N}_0}\,)\!] \\
\equiv & \quad \{\ \text{cancellations (ana and cata)}\ \} \\
& f = [\underline{1}, \mathsf{mul}] \cdot \mathbf{F}\ m \cdot \mathsf{out} \cdot \mathsf{in} \cdot \mathbf{F}\ k \cdot (id + \langle \mathsf{succ}, id \rangle) \cdot \mathsf{out}_{\mathbb{N}_0} \\
\equiv & \quad \{\ \mathsf{in} \cdot \mathsf{out} = id \text{ ; functor } \mathbf{F}\text{: } (\mathbf{F}\ m) \cdot (\mathbf{F}\ k) = \mathbf{F}\ (m \cdot k)\ \} \\
& f = [\underline{1}, \mathsf{mul}] \cdot \mathbf{F}\ (m \cdot k) \cdot (id + \langle \mathsf{succ}, id \rangle) \cdot \mathsf{out}_{\mathbb{N}_0} \\
\equiv & \quad \{\ \text{isomorphism } \mathsf{in}_{\mathbb{N}_0} \,/\, \mathsf{out}_{\mathbb{N}_0} ;\ m \cdot k = f \text{ ; } \mathbf{F}\ f = id + id \times f\ \} \\
& f \cdot \mathsf{in}_{\mathbb{N}_0} = [\underline{1}, \mathsf{mul}] \cdot (id + id \times f) \cdot (id + \langle \mathsf{succ}, id \rangle)
\end{aligned}
$$

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

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

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

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

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

$\equiv$ { go pointwise }

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

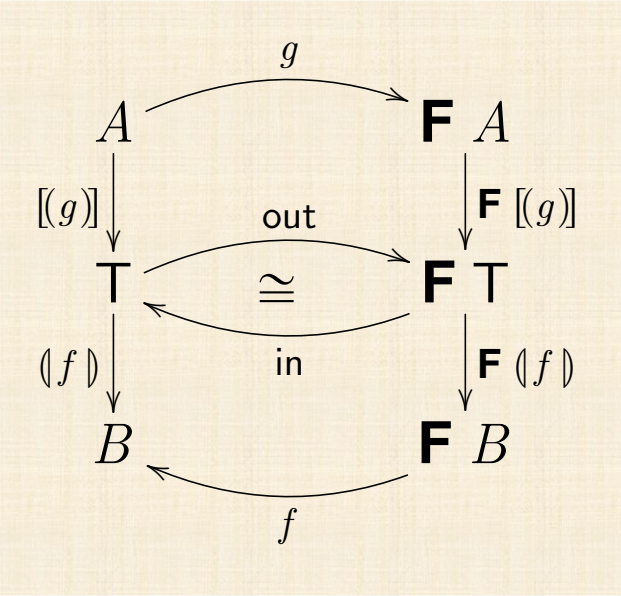
HILOMORPHISM

"Hylo + morphism"

ξύλο = matter, thing

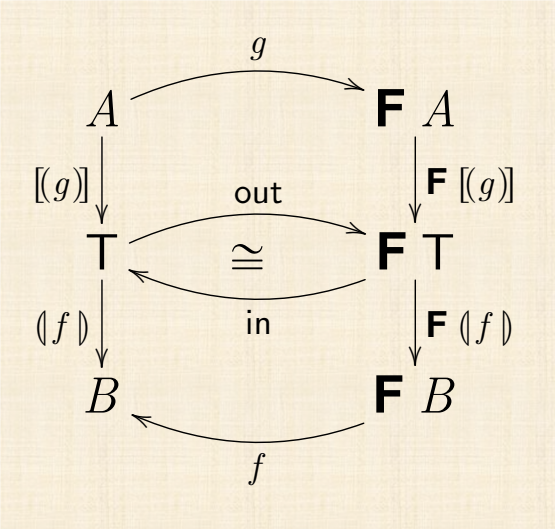
$$[\![f, g]\!] = (\!| f |\!) \cdot [\!(g)\!]$$

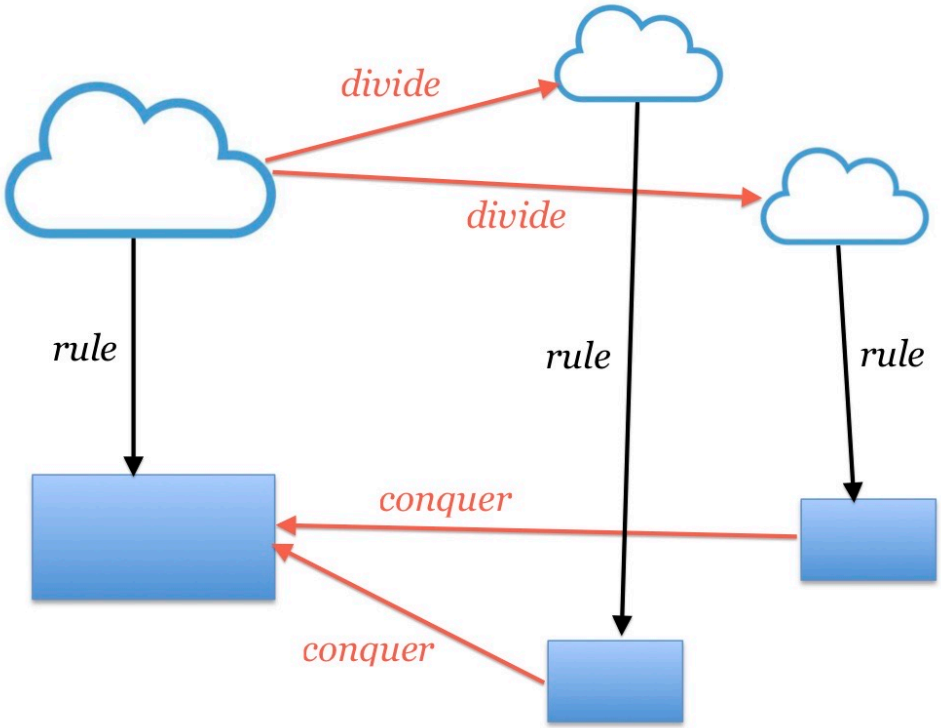
HILOMORPHISM



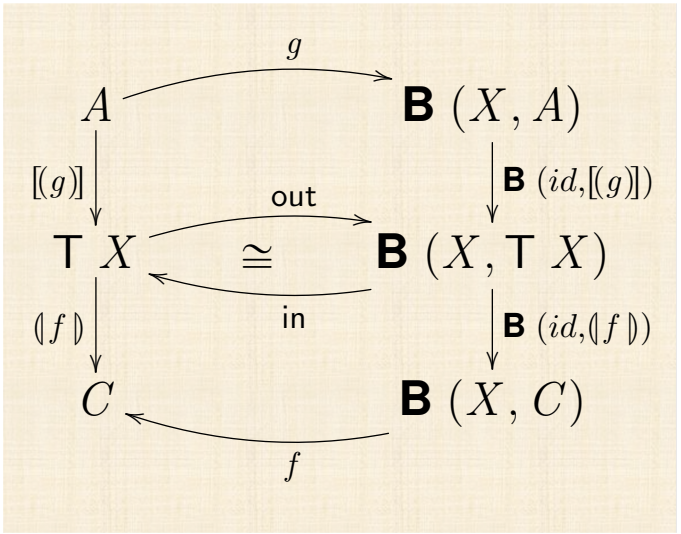
$$[\![f, g]\!] = (\!|\, f \,|\!) \cdot [\!(g)\!]$$

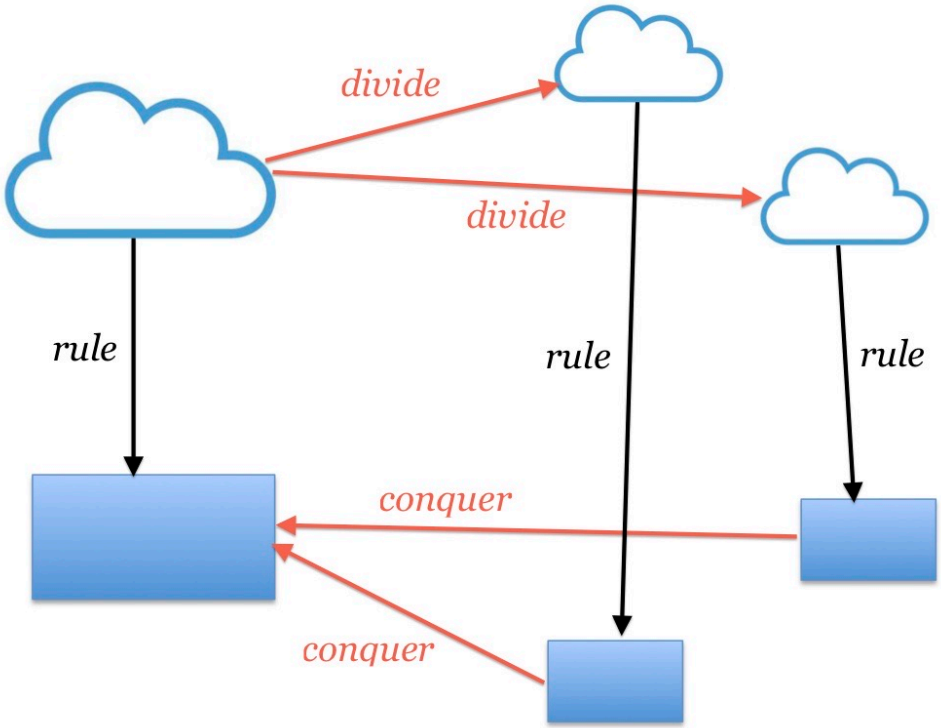
‘DIVIDE & CONQUER’





‘DIVIDE & CONQUER’

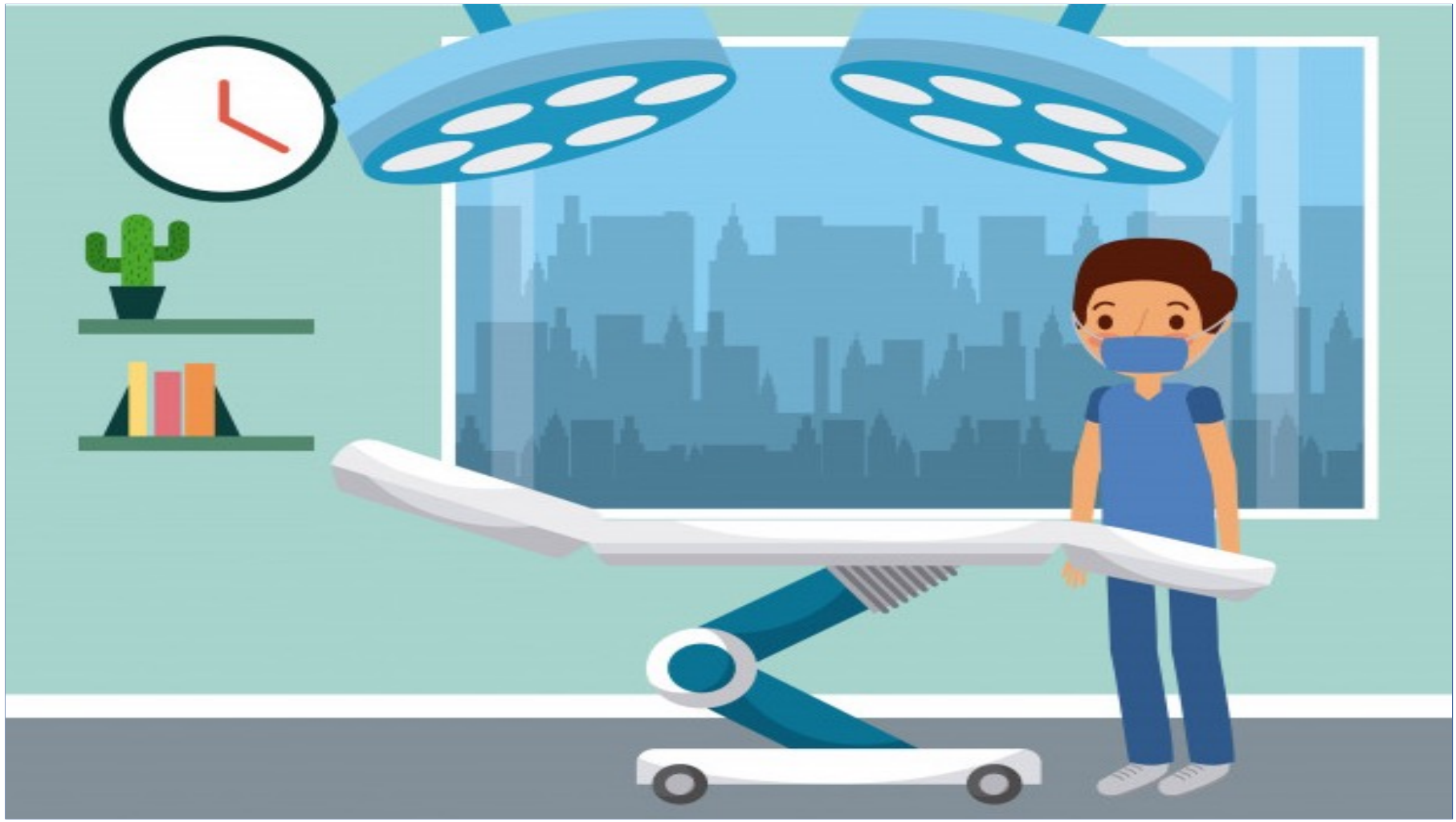


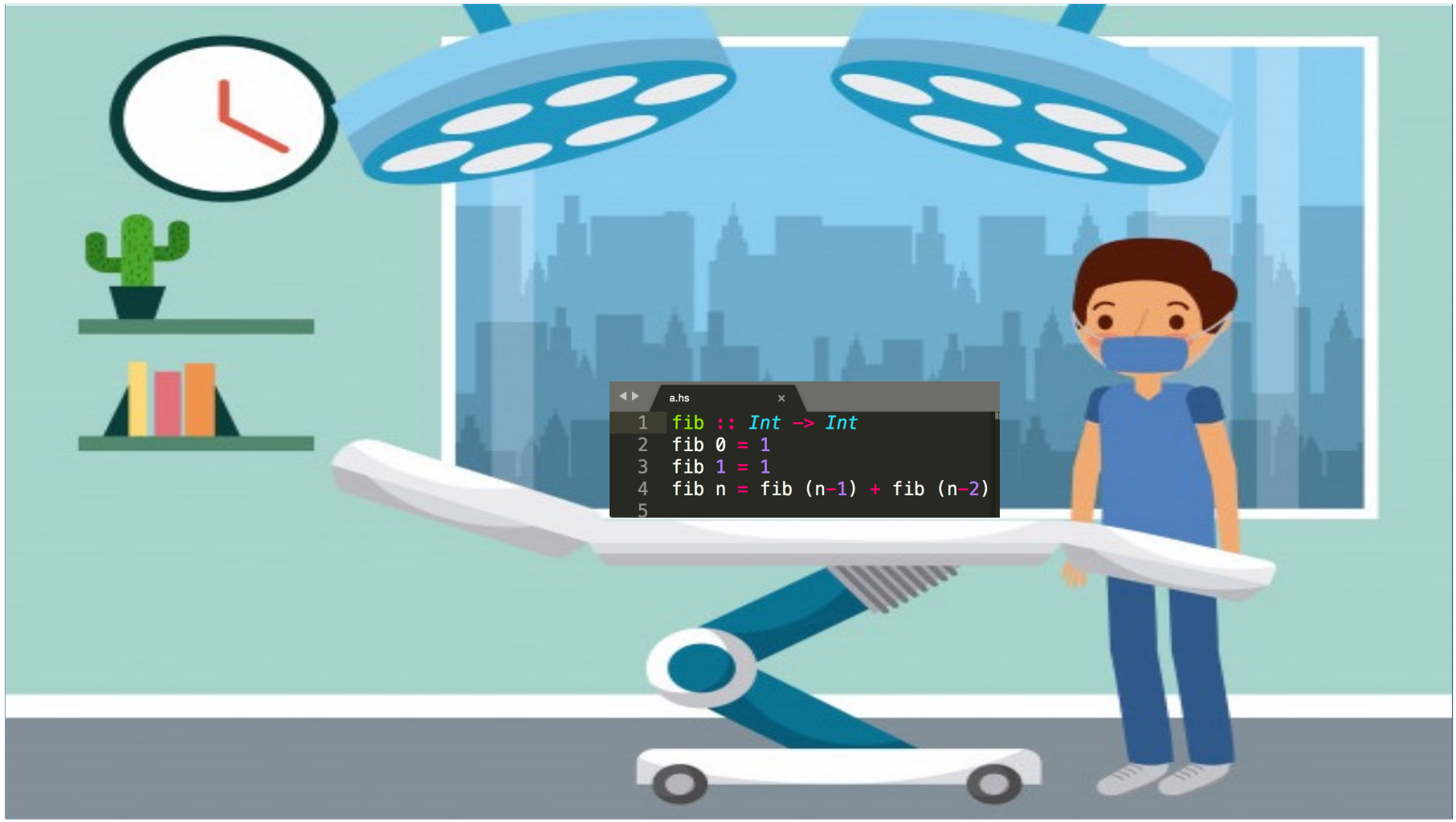


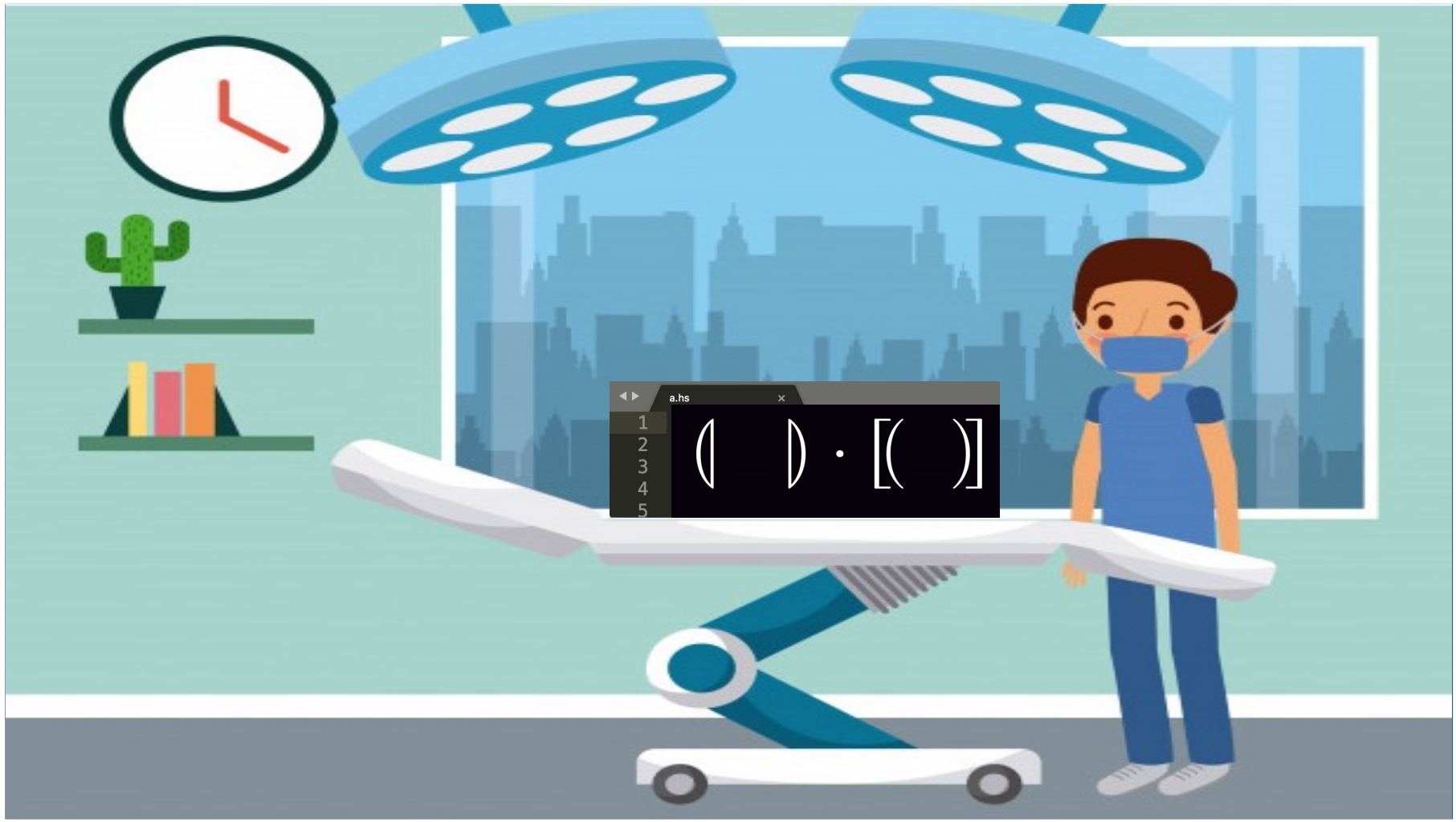
`DIVIDE & CONQUER’

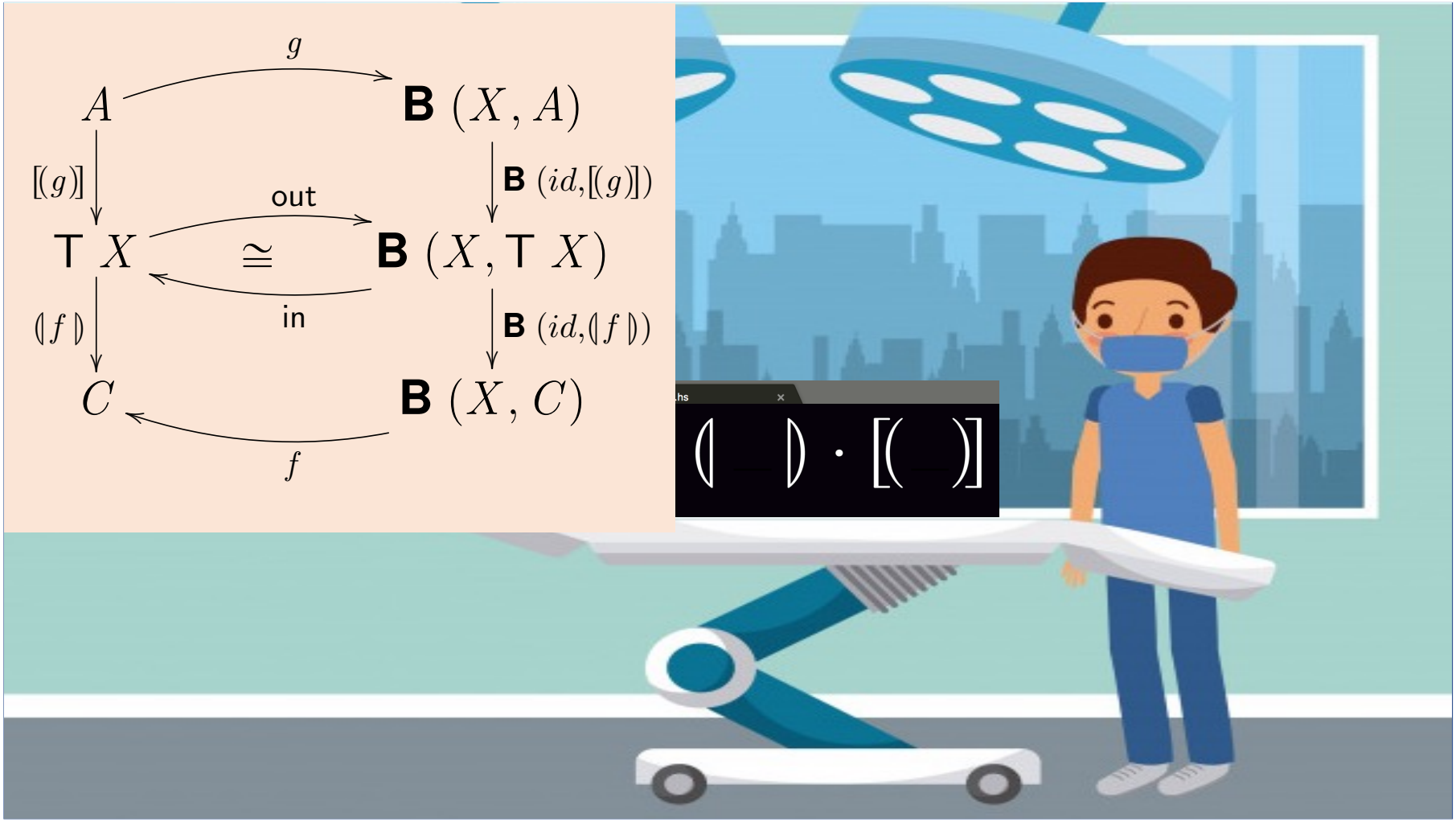
$$C \xleftarrow{(\!|-|\!)} \mathsf{T} \xleftarrow{[\![-]\!]} A$$

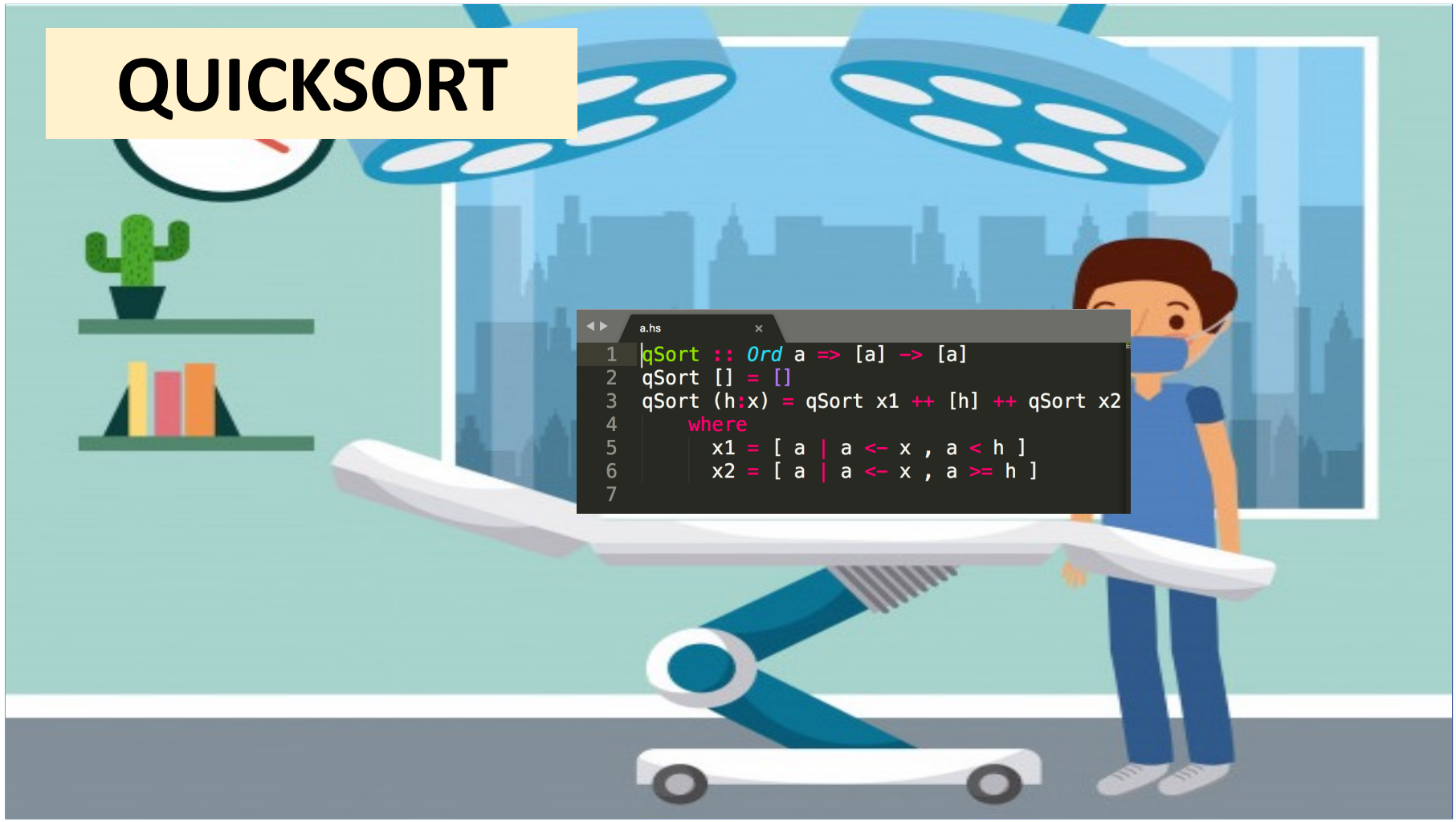
Cálculo de Programas
Aula T10(a)




a.hs
1 fib :: Int -> Int
2 fib 0 = 1
3 fib 1 = 1
4 fib n = fib (n-1) + fib (n-2)
5


a.hs
1
2
3
4
5
( ) · [( )]

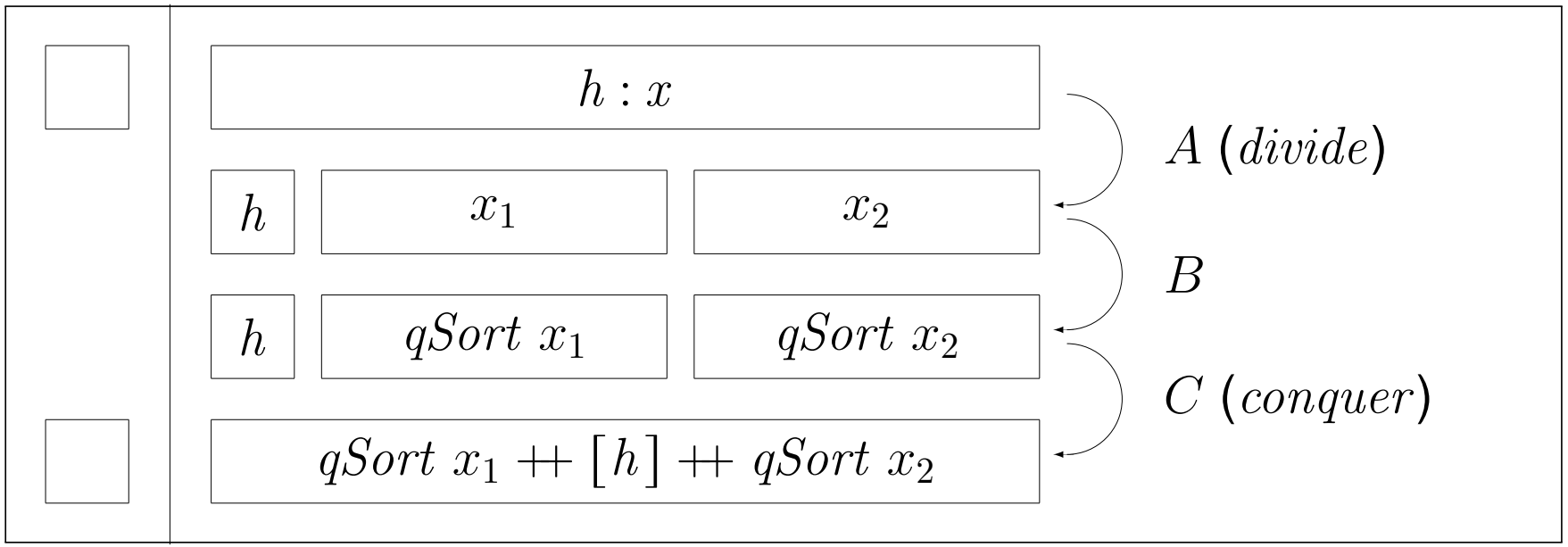

g
A
B (X, A)
[(g)]
B (id,[(g)])
out
T X
≅
B (X, T X)
in
(| f |)
B (id,(| f |))
C
B (X, C)
f
(| _ |) · [(_)]


QUICKSORT
a.hs
1 qSort :: Ord a => [a] -> [a]
2 qSort [] = []
3 qSort (h:x) = qSort x1 ++ [h] ++ qSort x2
4 where
5 x1 = [a | a <- x , a < h]
6 x2 = [a | a <- x , a >= h]
7

QUICKSORT

a.hs

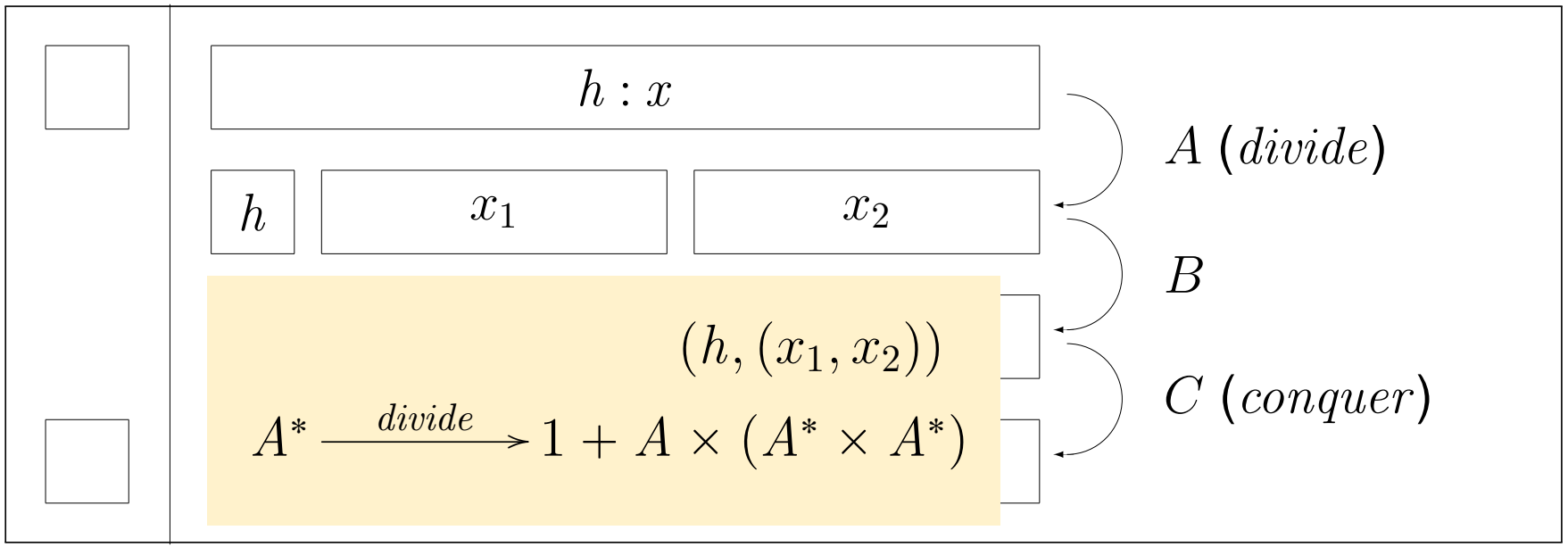
```haskell
qSort :: Ord a => [a] -> [a]
qSort [] = []
qSort (h:x) = qSort x1 ++ [h] ++ qSort x2
    where
      x1 = [ a | a <- x , a < h ]
      x2 = [ a | a <- x , a >= h ]

```



QUICKSORT

a.hs

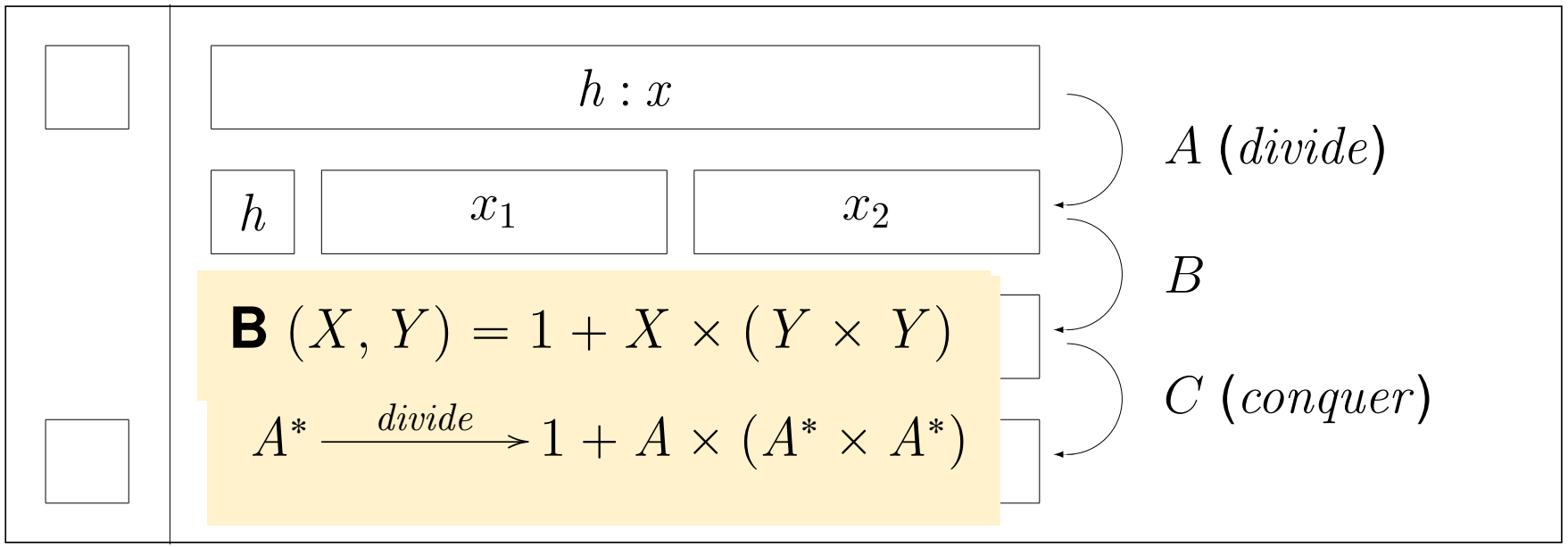
```haskell
qSort :: Ord a => [a] -> [a]
qSort [] = []
qSort (h:x) = qSort x1 ++ [h] ++ qSort x2
    where
      x1 = [ a | a <- x , a < h ]
      x2 = [ a | a <- x , a >= h ]

```



QUICKSORT

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 ]

```

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

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

$$g_2\ (h, x) = (h, (x_1, x_2))$$
$$\textbf{where}$$
$$x_1 = [a \mid a \leftarrow x, a < h]$$
$$x_2 = [a \mid a \leftarrow x, a \geqslant h]$$

QUICKSORT

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

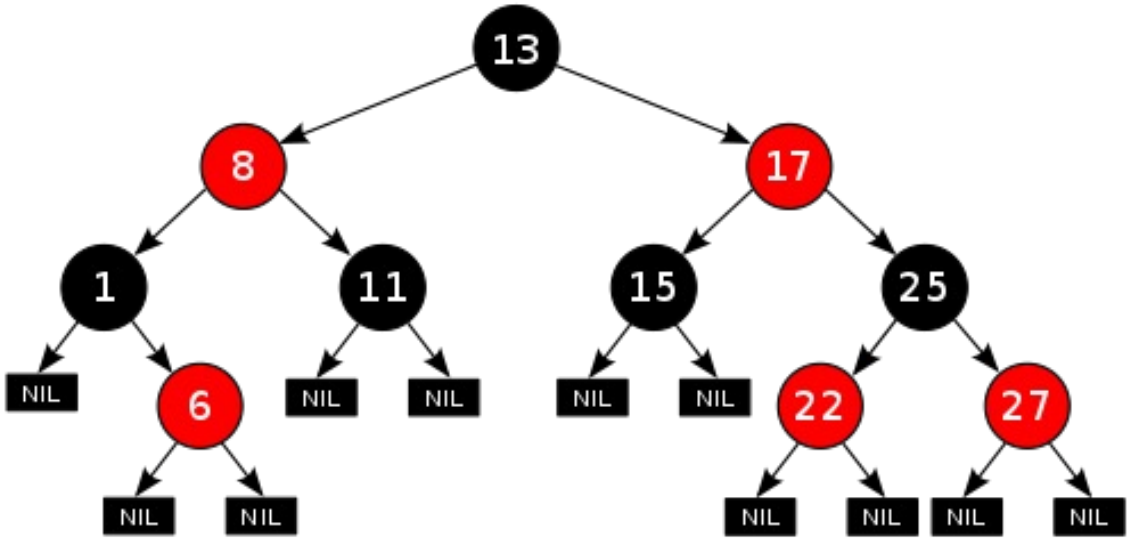
$$\begin{cases} qSort \cdot \mathsf{nil} = \mathsf{nil} \\ qSort \cdot \mathsf{cons} = f_2 \cdot (id \times (qSort \times qSort)) \cdot g_2 \end{cases}$$

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

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

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

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




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

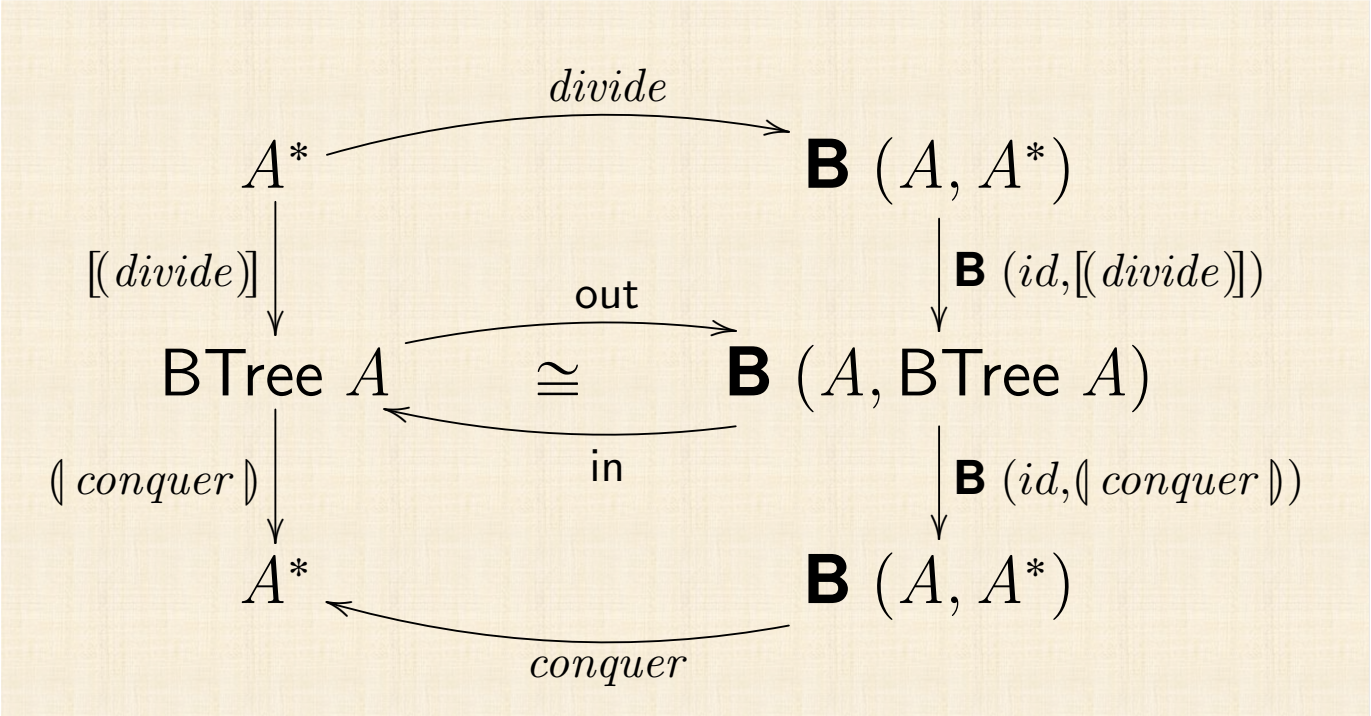
QUICKSORT

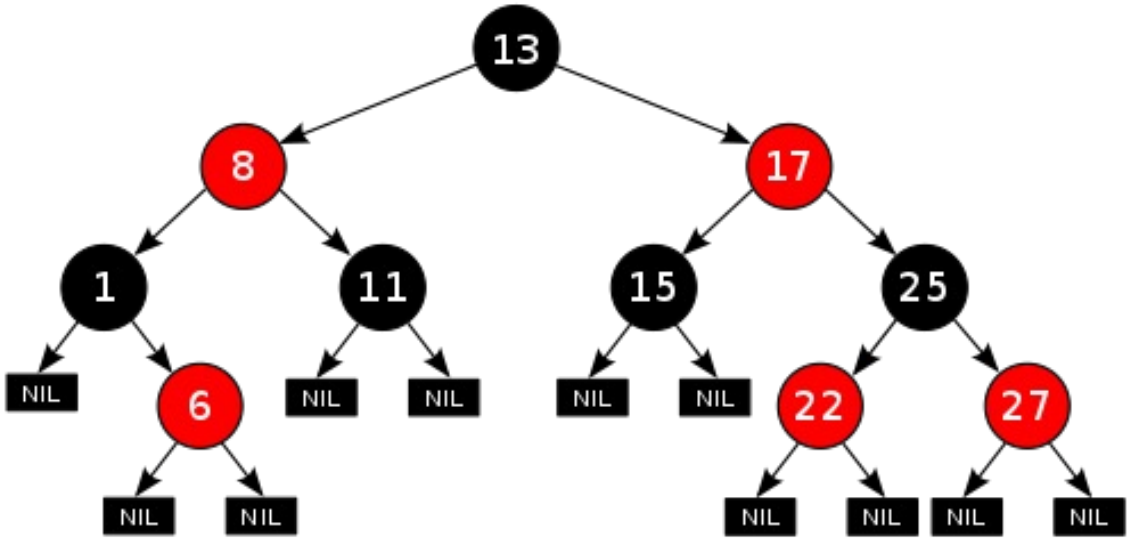
$$\mathbf{B}\,(X, Y) = 1 + X \times (Y \times Y)$$

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	$X + X \times Y$	$id + id \times f$	$f + f \times id$
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

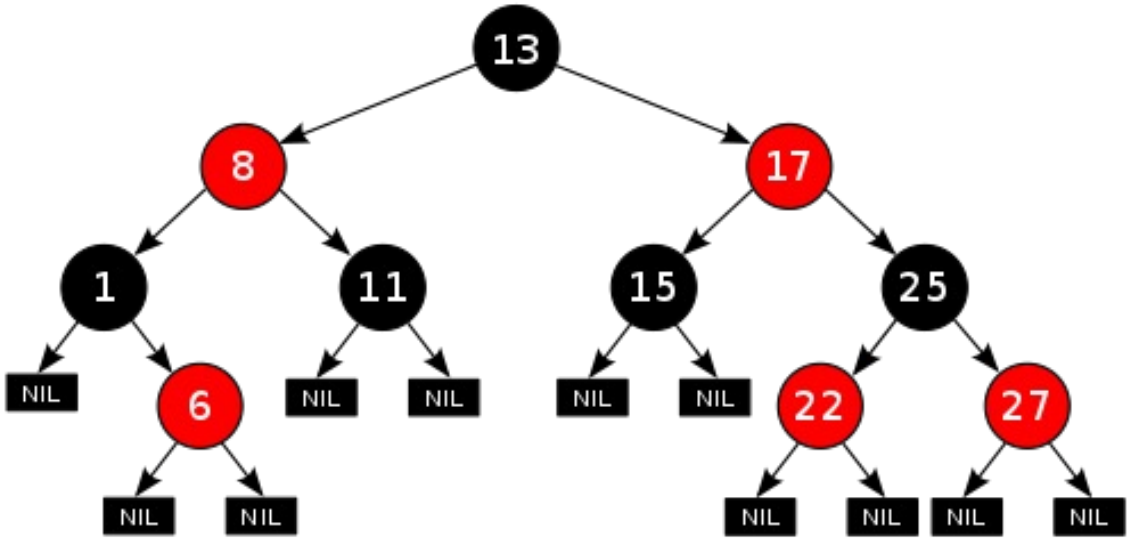
```haskell
qSort :: Ord a => [a] -> [a]
qSort [] = []
qSort (h:x) = qSort x1 ++ [h] ++ qSort x2
    where
      x1 = [ a | a <- x , a < h ]
      x2 = [ a | a <- x , a >= h ]
```



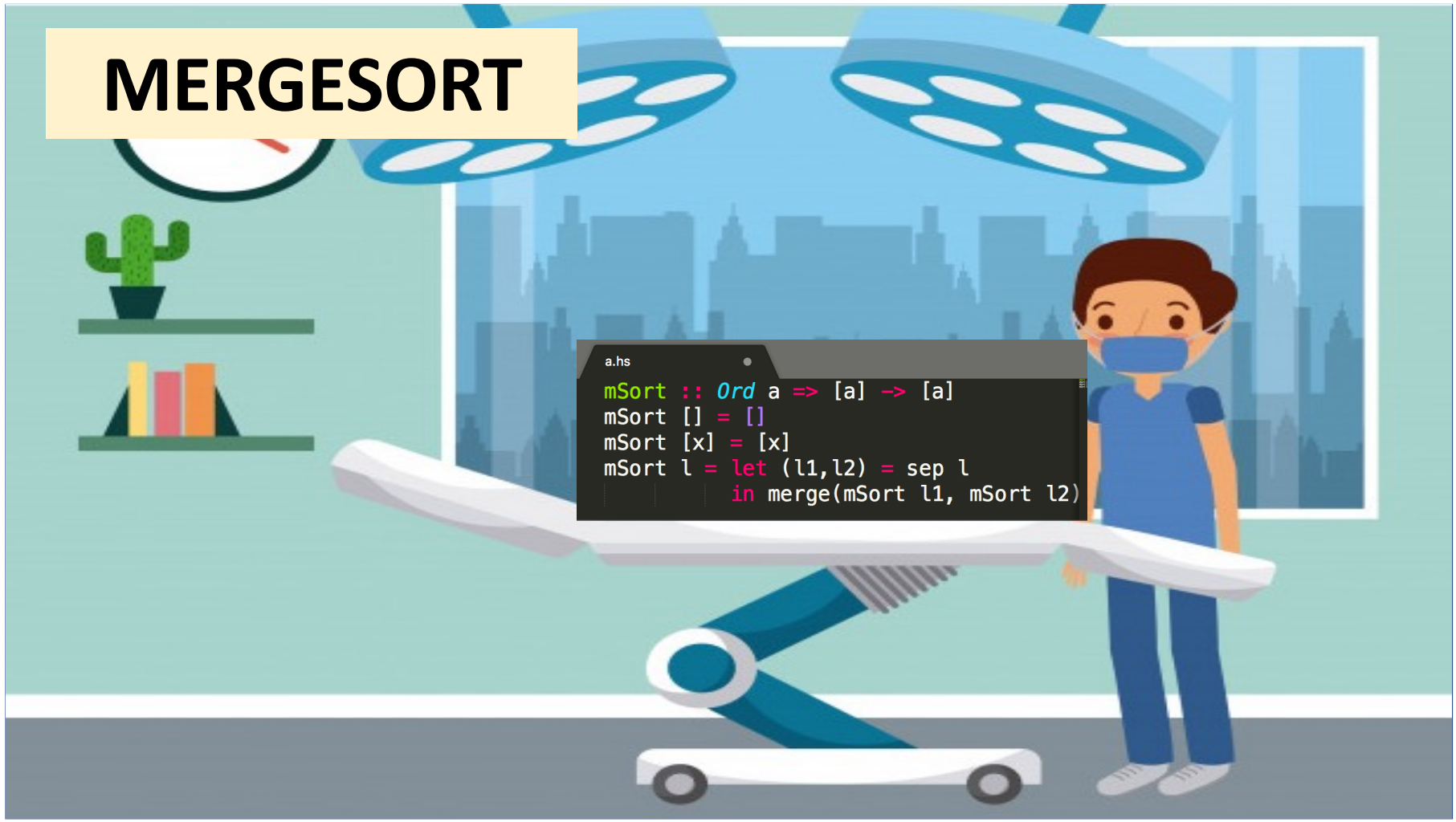



```
t = anaB divide [13,8,17,1,6,11,25,15,27,22]
```

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



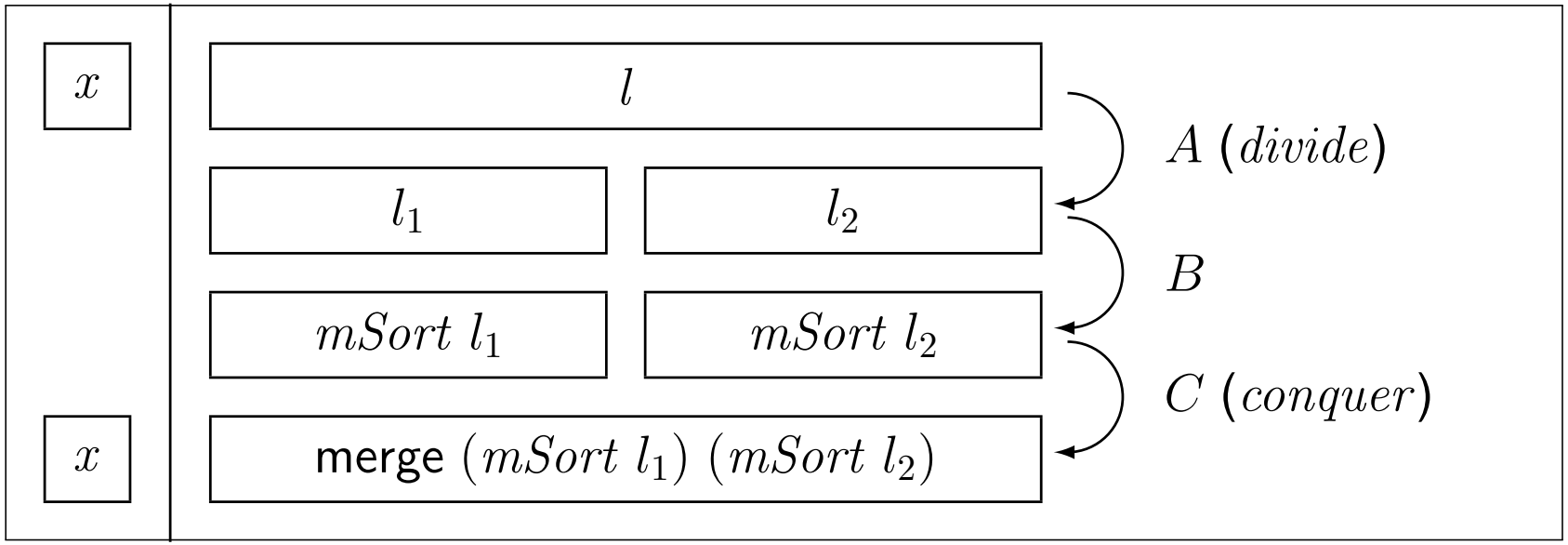

```
*Cp> anaB divide [13,8,17,1,6,11,25,15,27,22]
Node (13,(Node (8,(Node (1,(Empty,Node (6,(Empty,Empty)))),Node
(11,(Empty,Empty)))),Node (17,(Node (15,(Empty,Empty)),Node (25,
(Node (22,(Empty,Empty)),Node (27,(Empty,Empty)))))))
```


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

a.hs

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

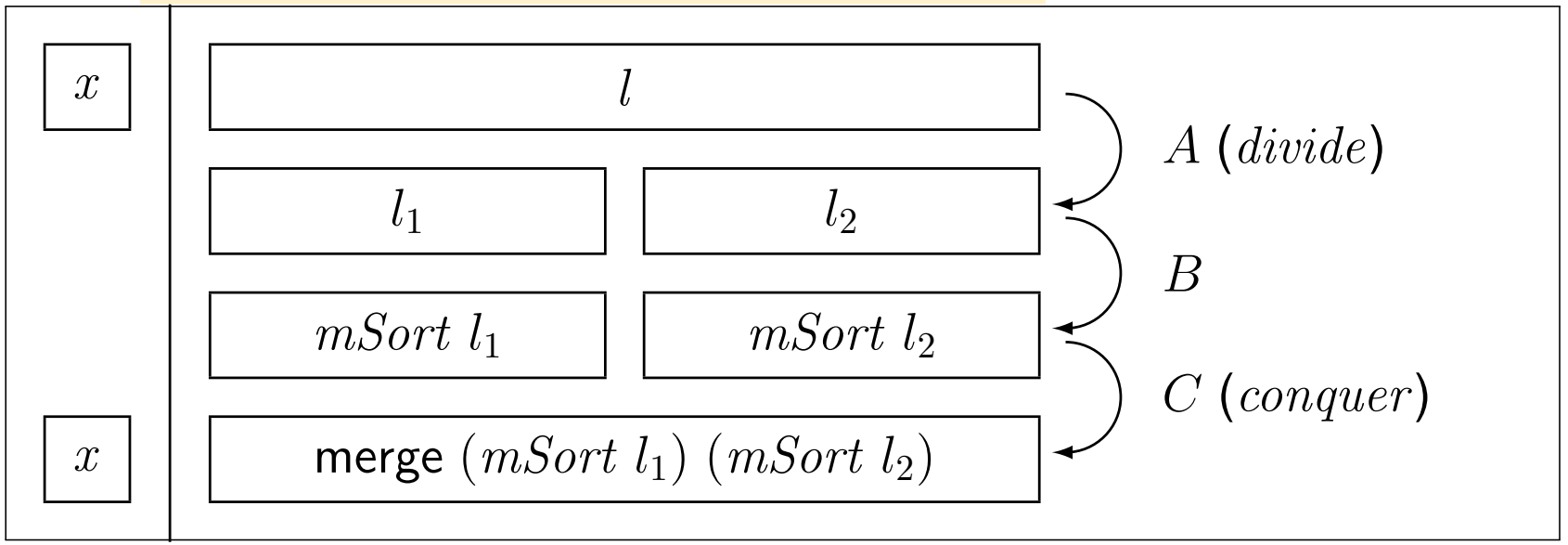


MERGESORT

a.hs

```
mSort :: Ord a => [a] -> [a]


l1,l2) = sep l
rge(mSort l1, mSort l2)
```

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

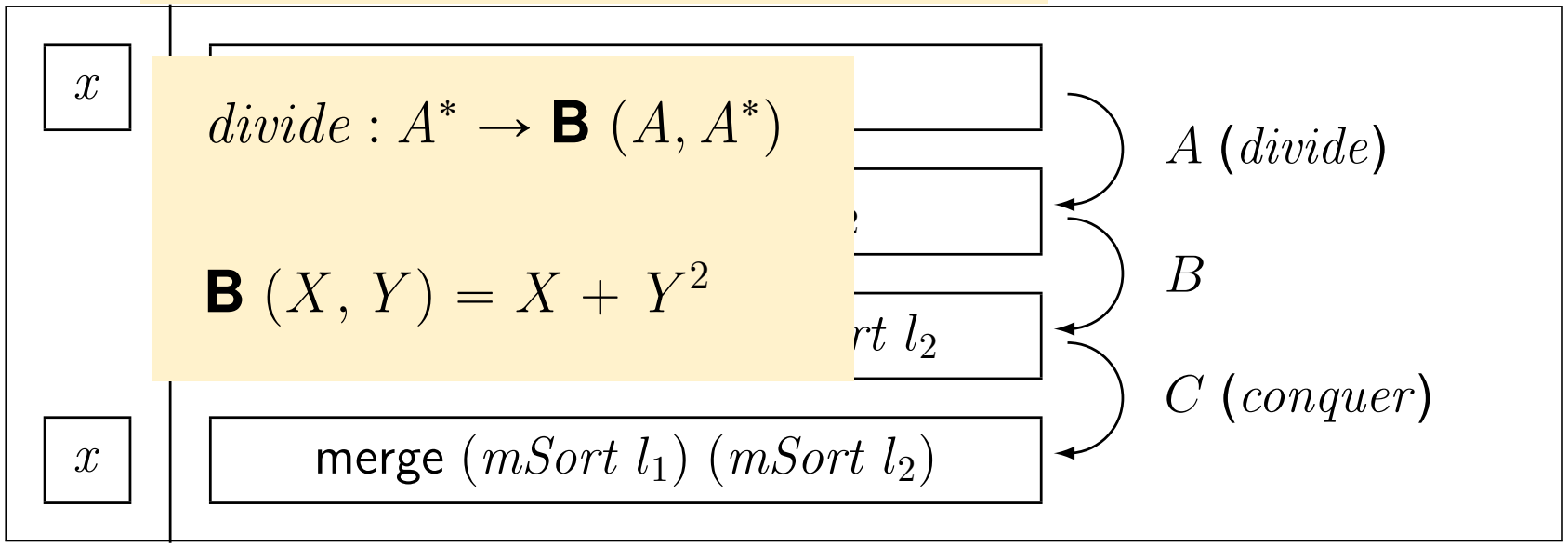


MERGESORT

a.hs

```
mSort :: Ord a => [a] -> [a]


l1,l2) = sep l
rge(mSort l1, mSort l2)
```

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



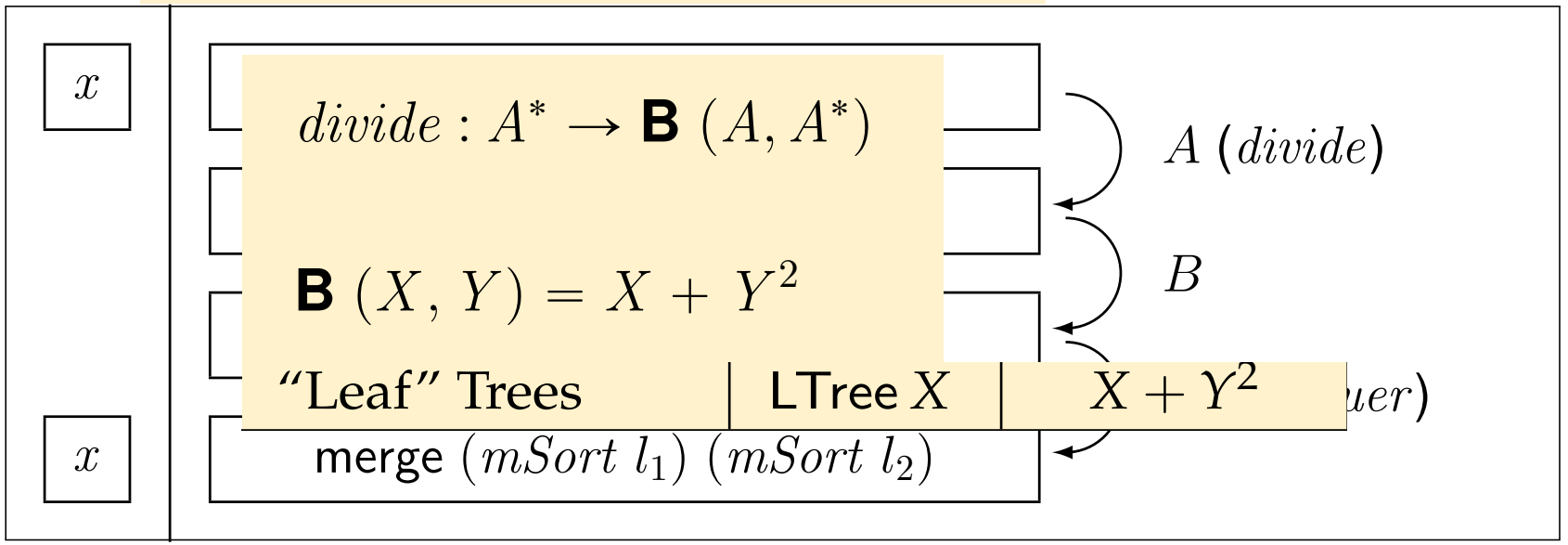
$$divide : A^* \rightarrow \mathbf{B}\ (A, A^*)$$

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

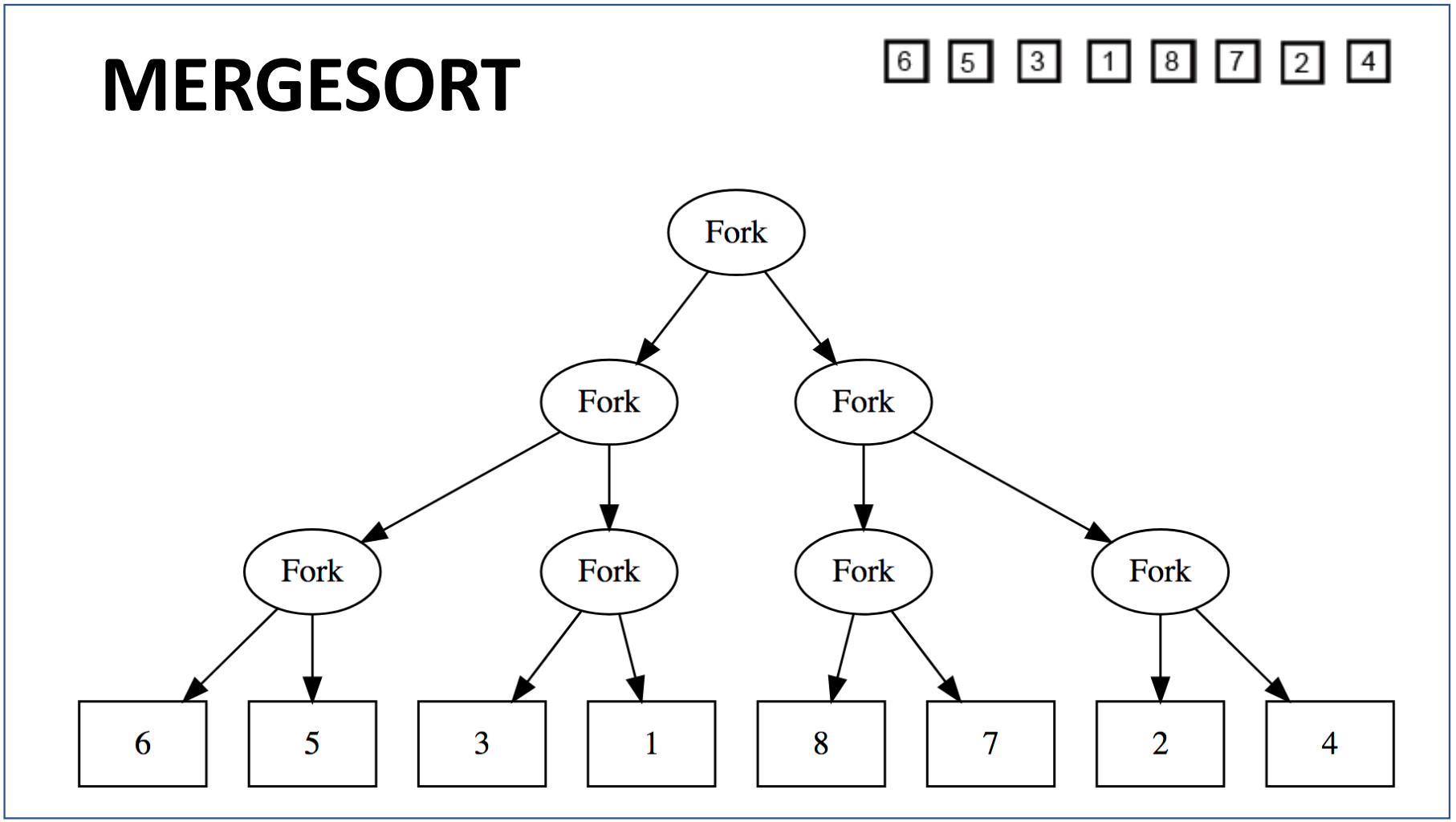
MERGESORT

```
a.hs
mSort :: Ord a => [a] -> [a]


l1,l2) = sep l
rge(mSort l1, mSort l2)
```

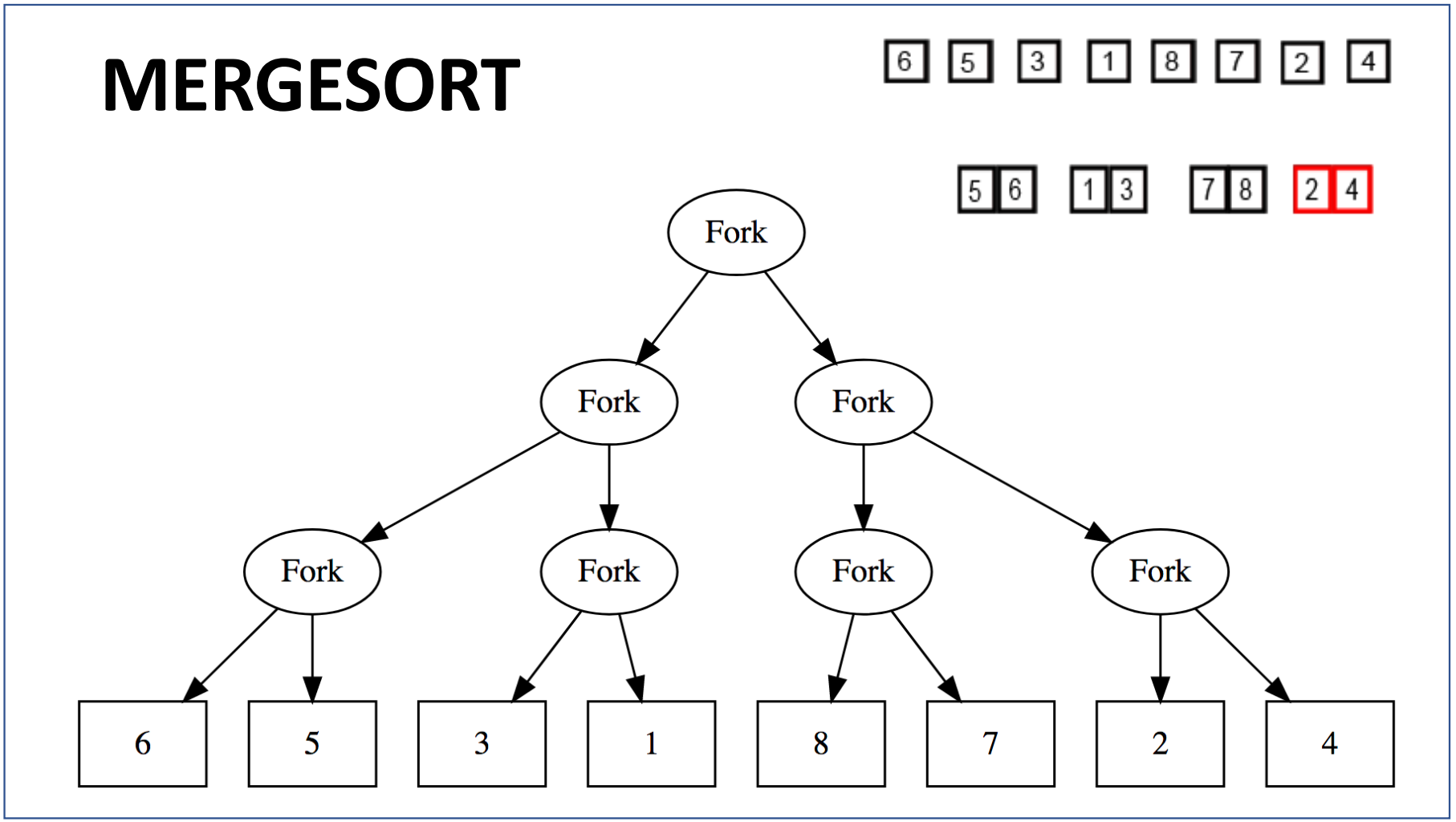
$$divide : A^* \rightarrow A + (A^* \times A^*)$$



MERGESORT



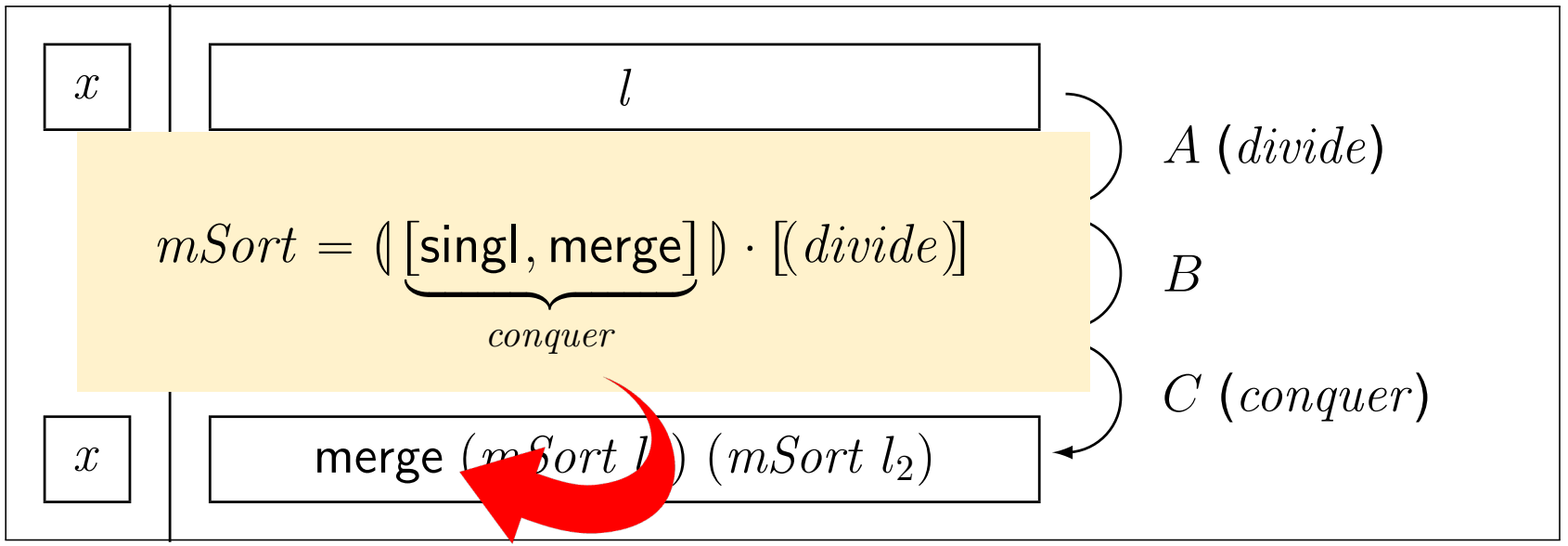
MERGESORT


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

a.hs

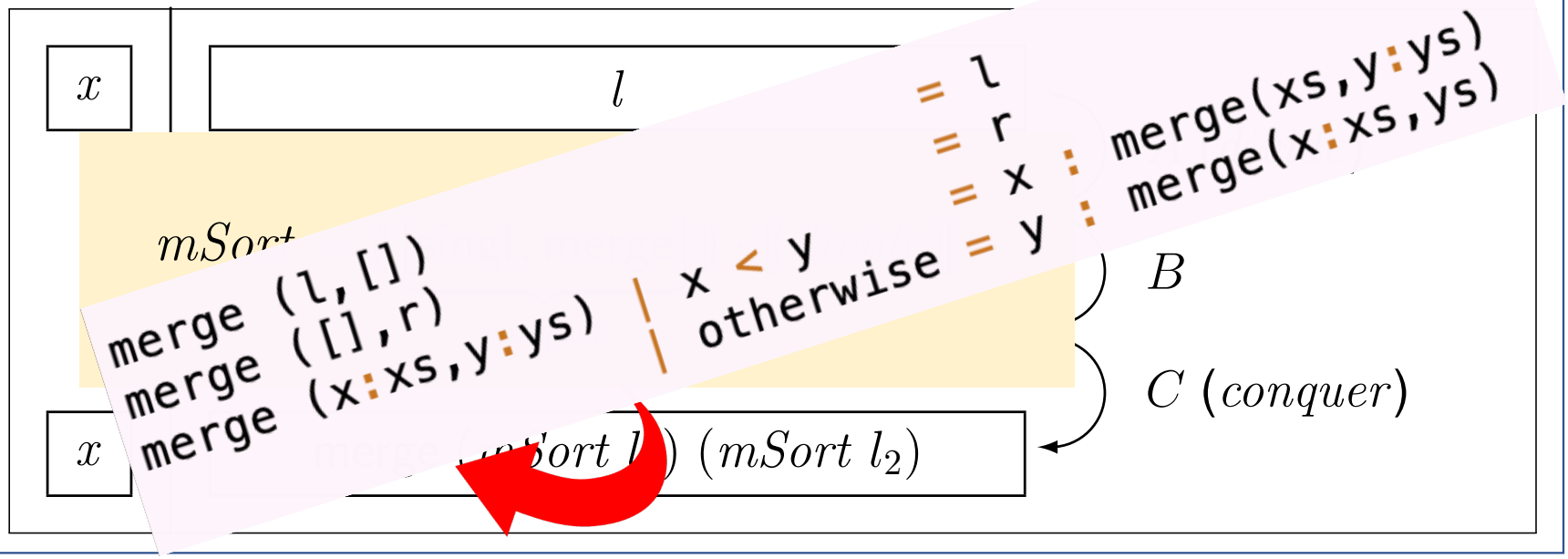
```
mSort :: Ord a => [a] -> [a]
mSort [] = []
mSort [x] = [x]
mSort l = let (l1,l2) = sep l
          in merge(mSort l1, mSort l2)
```



MERGESORT

a.hs

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

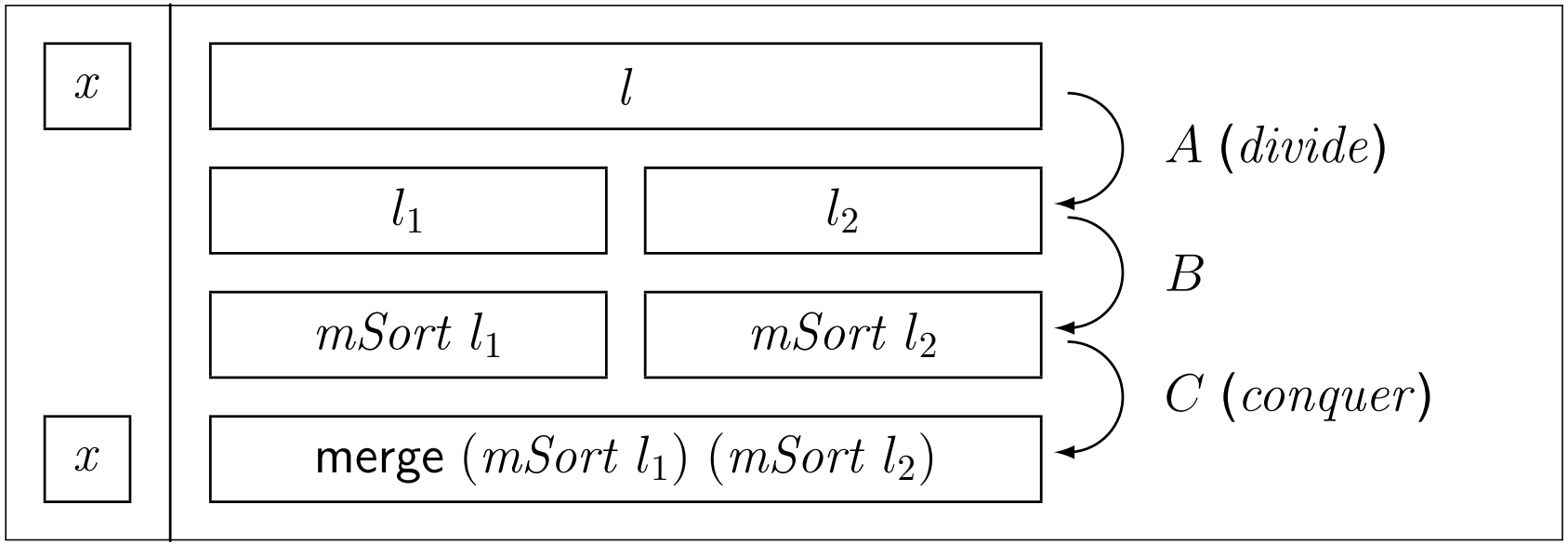



```
merge (l,[]) = l
merge ([],r) = r
merge (x:xs,y:ys) | x < y = x : merge(xs,y:ys)
                  | otherwise = y : merge(x:xs,ys)
```

MERGESORT

a.hs

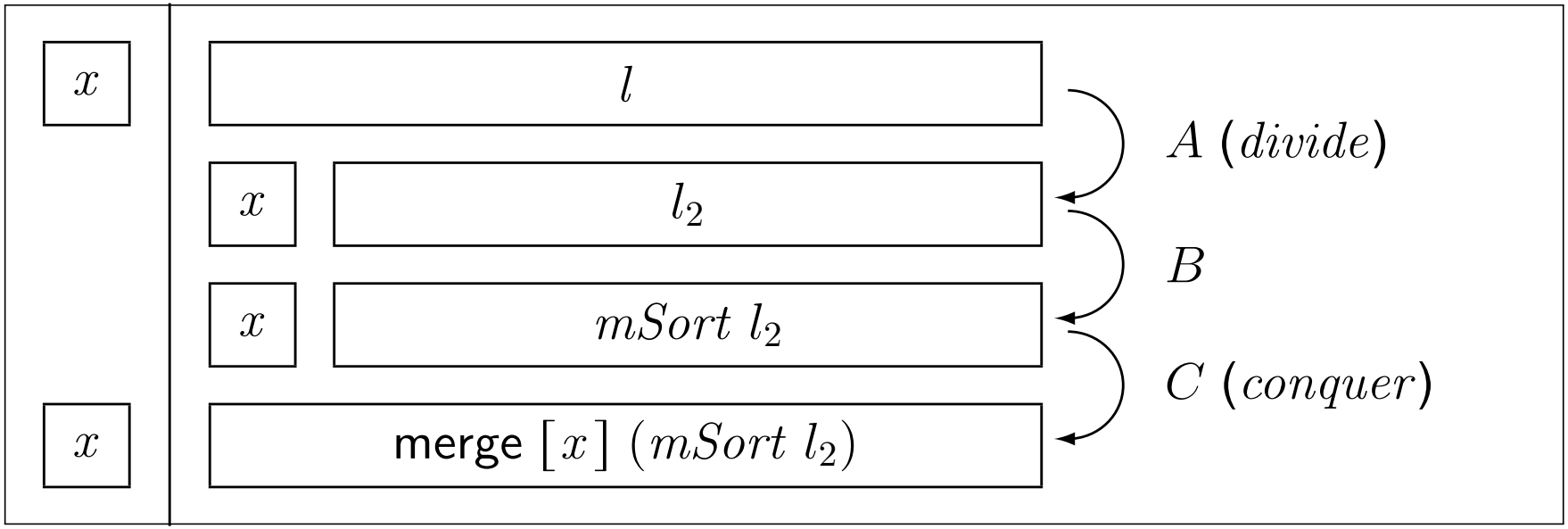
```
mSort :: Ord a => [a] -> [a]
mSort [] = []
mSort [x] = [x]
mSort l = let (l1,l2) = sep l
          in merge(mSort l1, mSort l2)
```



MERGESORT

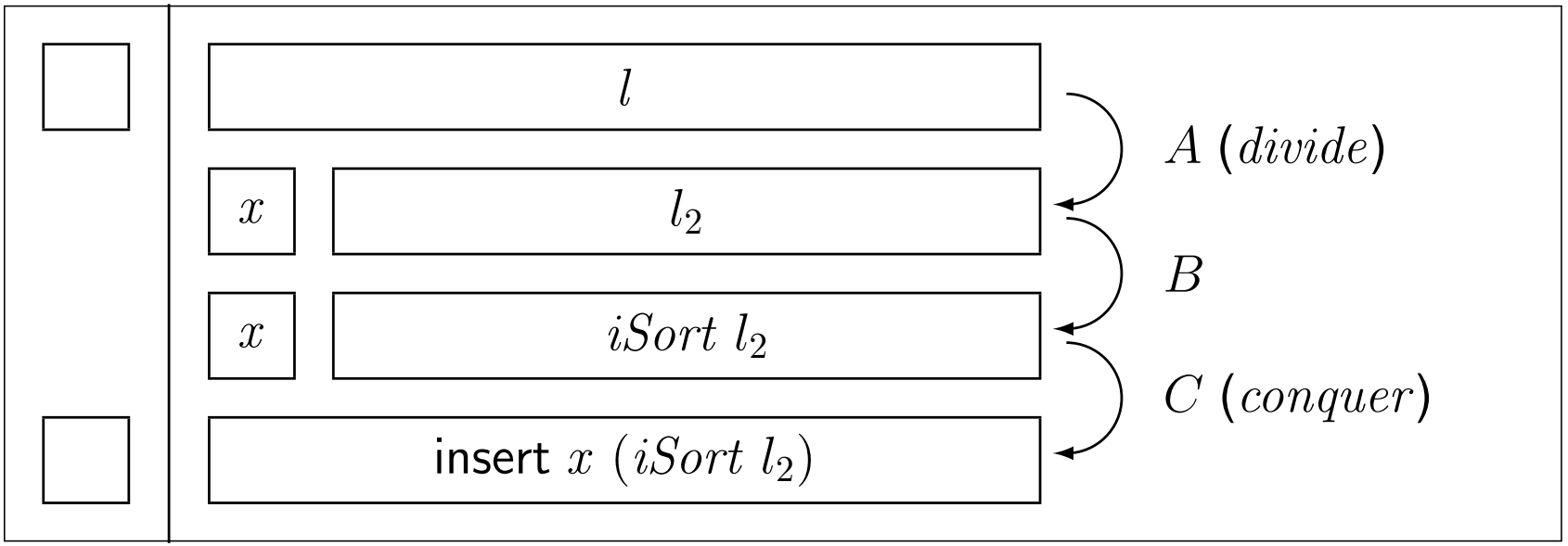
Particular case:

```
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
```



INSERTION SORT

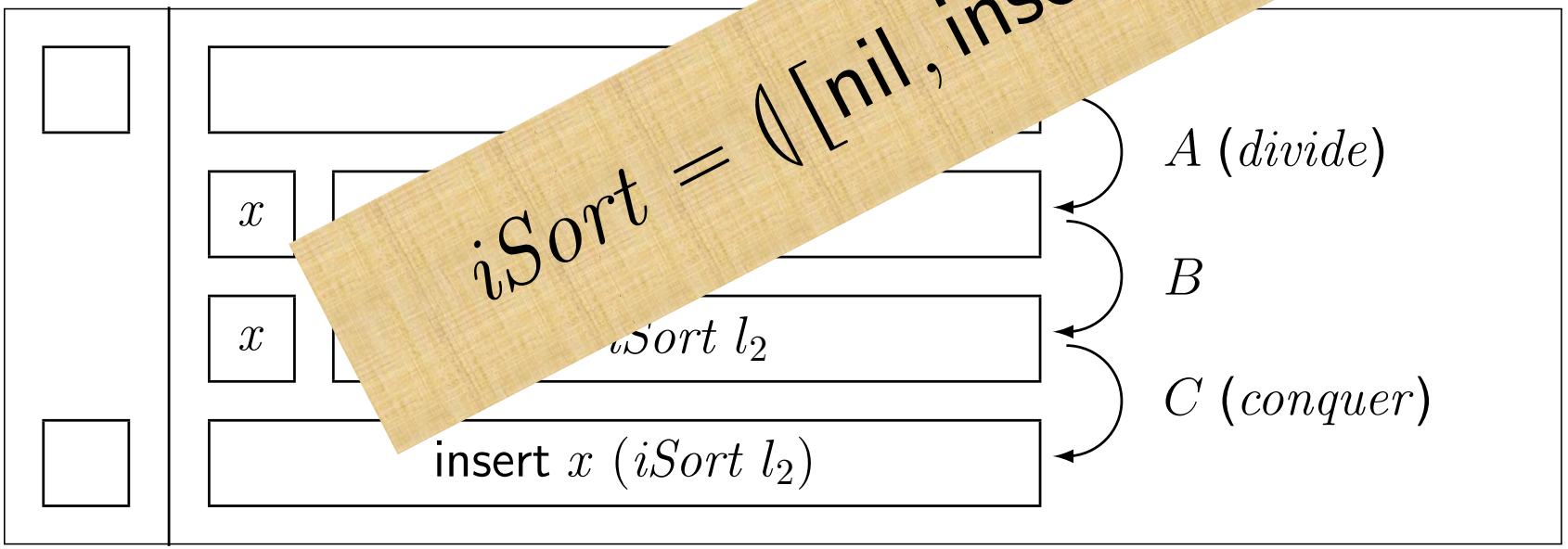
```haskell
insert :: Ord t => t -> [t] -> [t]
insert x []                     = [x]
insert x (y:ys) | x < y         = x:y:ys
                | otherwise = y:insert x ys
```



INSERTION SORT

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

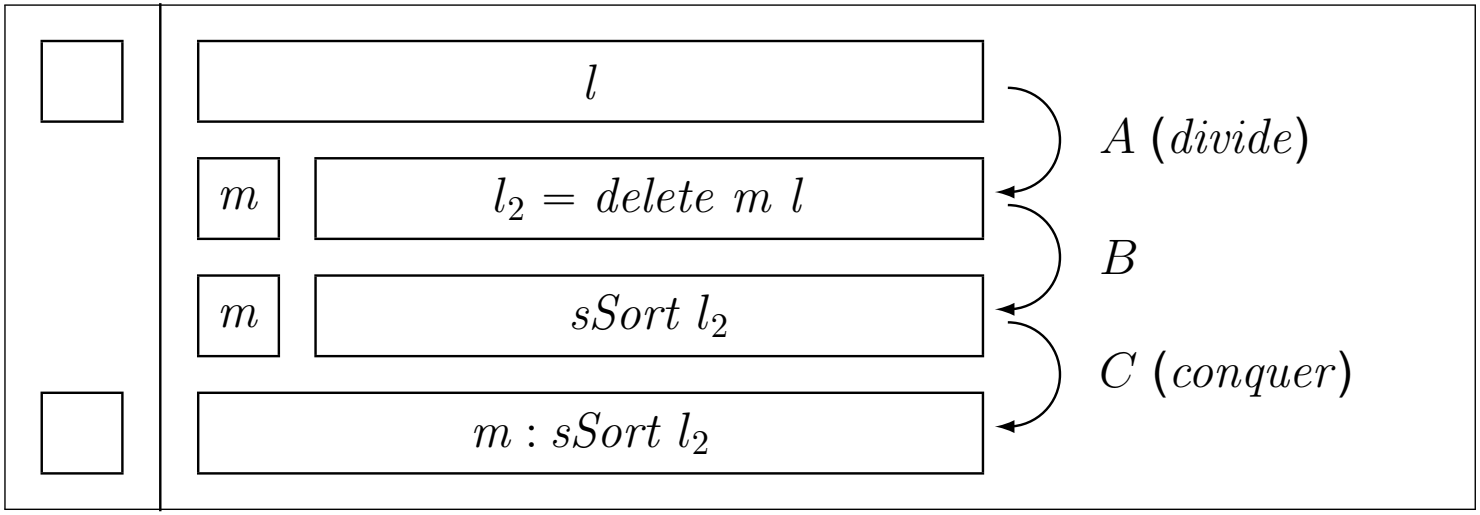
$$iSort = (\!|[nil, \widehat{insert}]|\!)$$



SELECTION SORT

```haskell
1
2  sSort :: Ord a => [a] -> [a]
3  sSort = anaL divide where
4     divide [] = i1()
5     divide (xs) = let m = minimum xs
6                   in  i2(m, delete m xs)
```

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



SELECTION SORT

```haskell
1
2  sSort :: Ord a => [a] -> [a]
3  sSort = anaL divide where
4     divide [] = i1()
5     divide (xs) = let m = minimum xs
6                   in  i2(m, delete m xs)
```

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

ANA, CATA & HILO

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

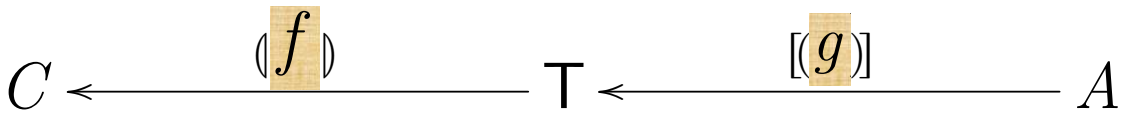
$$[\![f, g]\!] = (\!|f|\!) \cdot [\!(g)\!]$$

ANA, CATA & HILO

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

Reflexion laws:

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

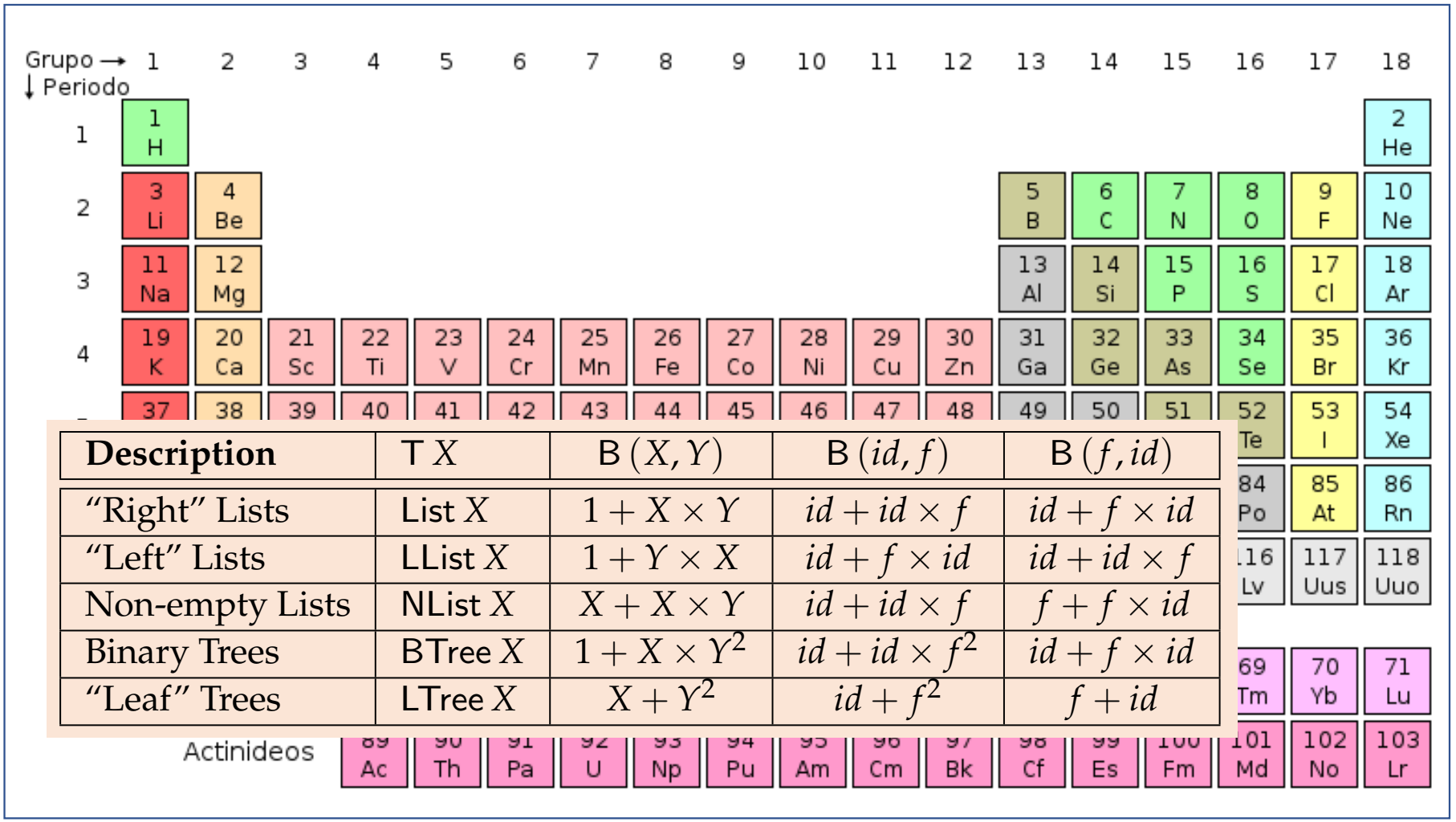


$$[\![f, g]\!] = (\!|\, f \,|\!) \cdot [\!(g)\!]$$

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*



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	$X + Y^2$	$id + f^2$	$f + id$

“Tabela periódica”

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 ]

```

		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	

Cálculo de Programas
Aula T10(b)

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.

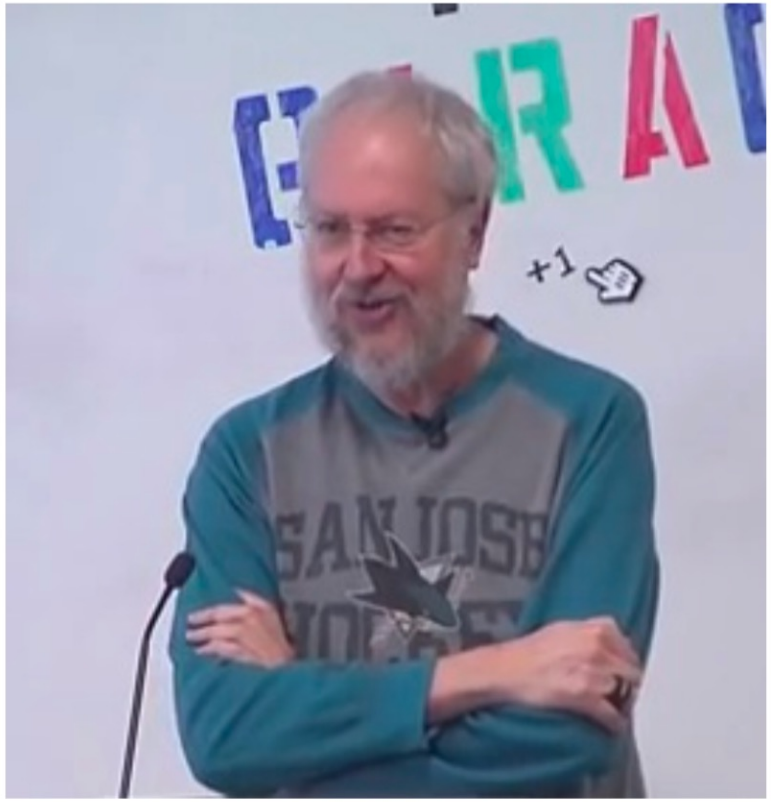
Our motivation to this concept will start from a well-known problem in functional programming (and computing as a whole) — that of coping with undefined computations.


CHANCE

Probability of the sum

"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)



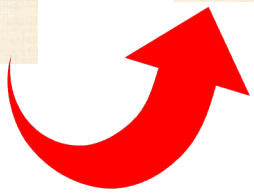
Douglas Crockford (2013)

Partial functions

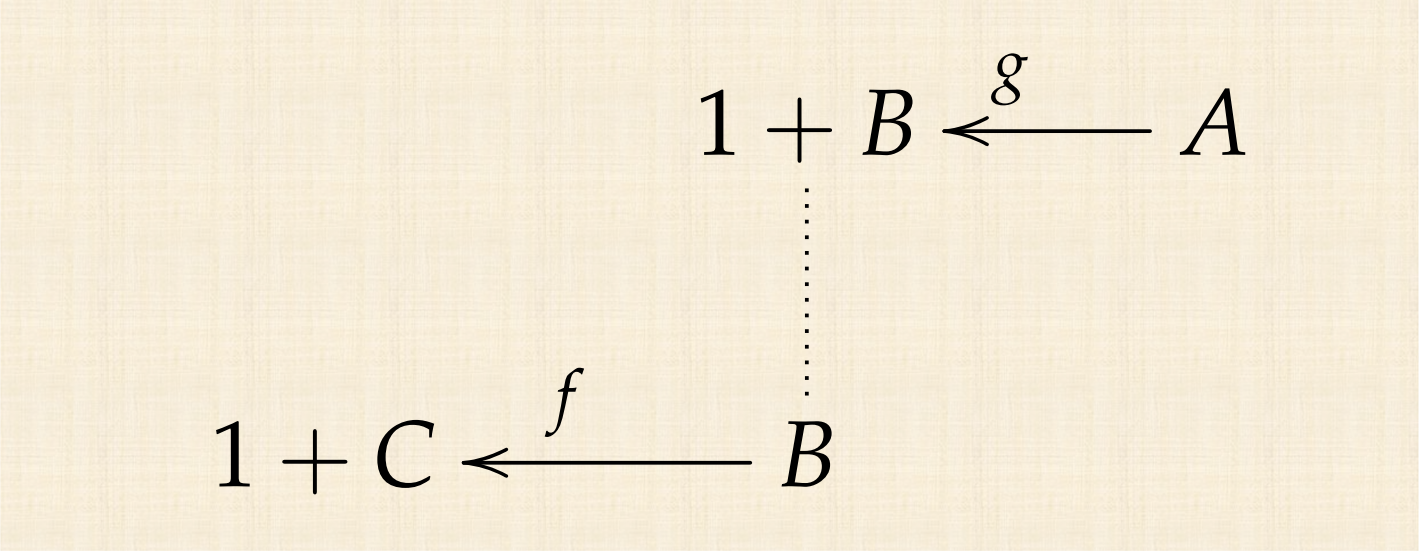
$$1 + B \xleftarrow{g} A$$

$$B \xleftarrow{f} A$$

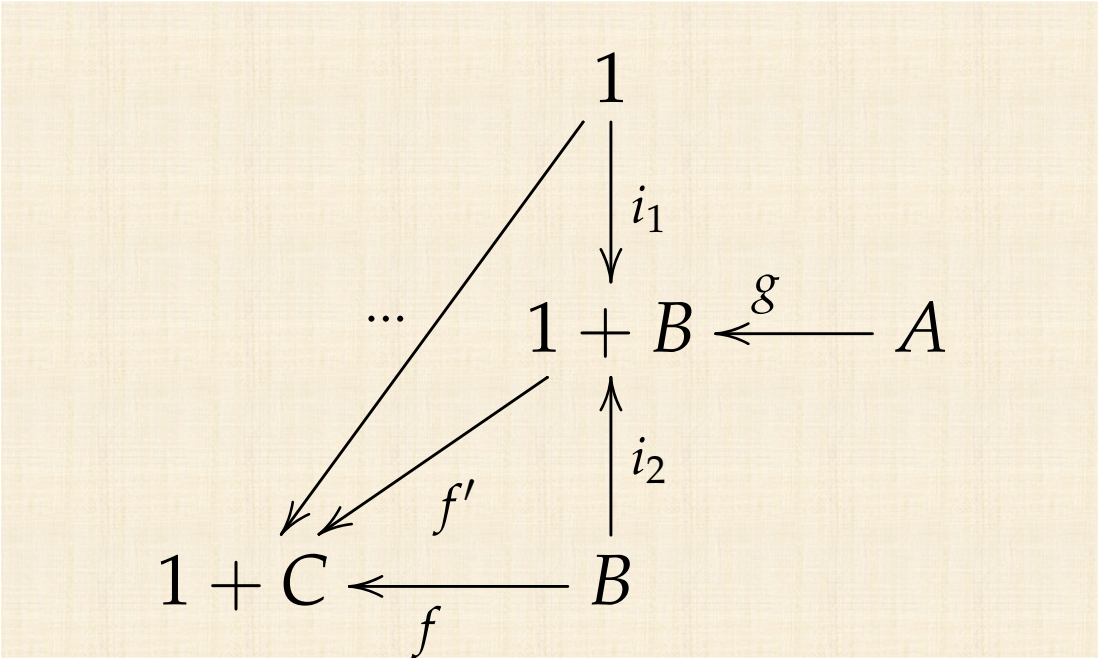
$$1 + B \xleftarrow{i_2 \cdot f} A$$



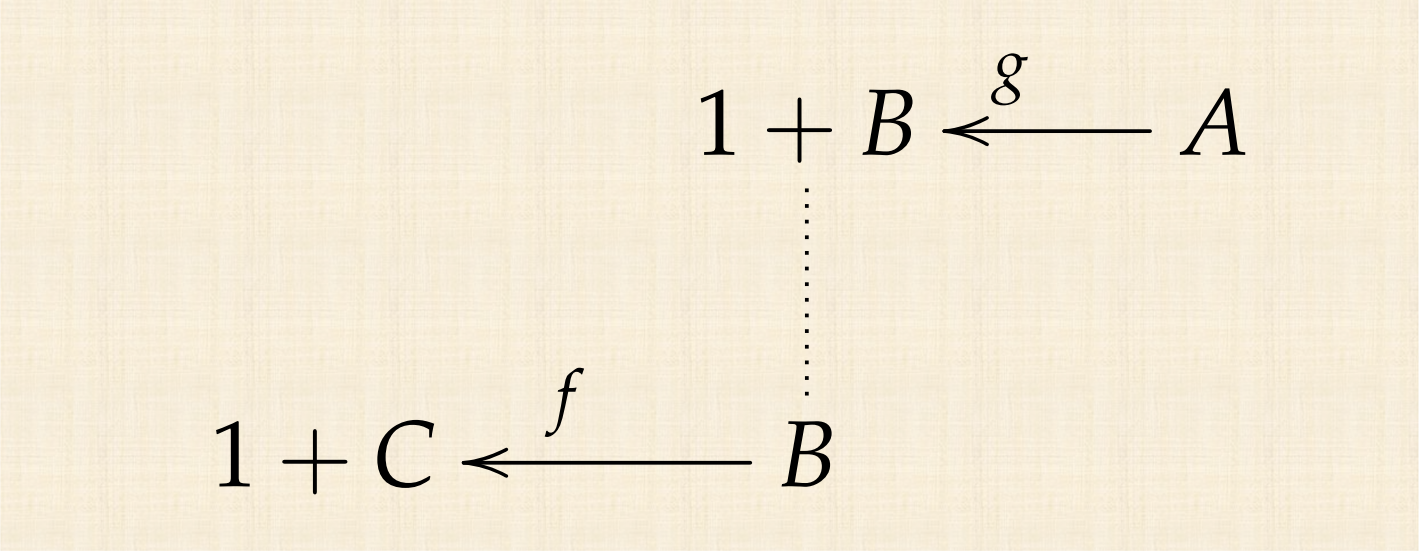
Partial functions



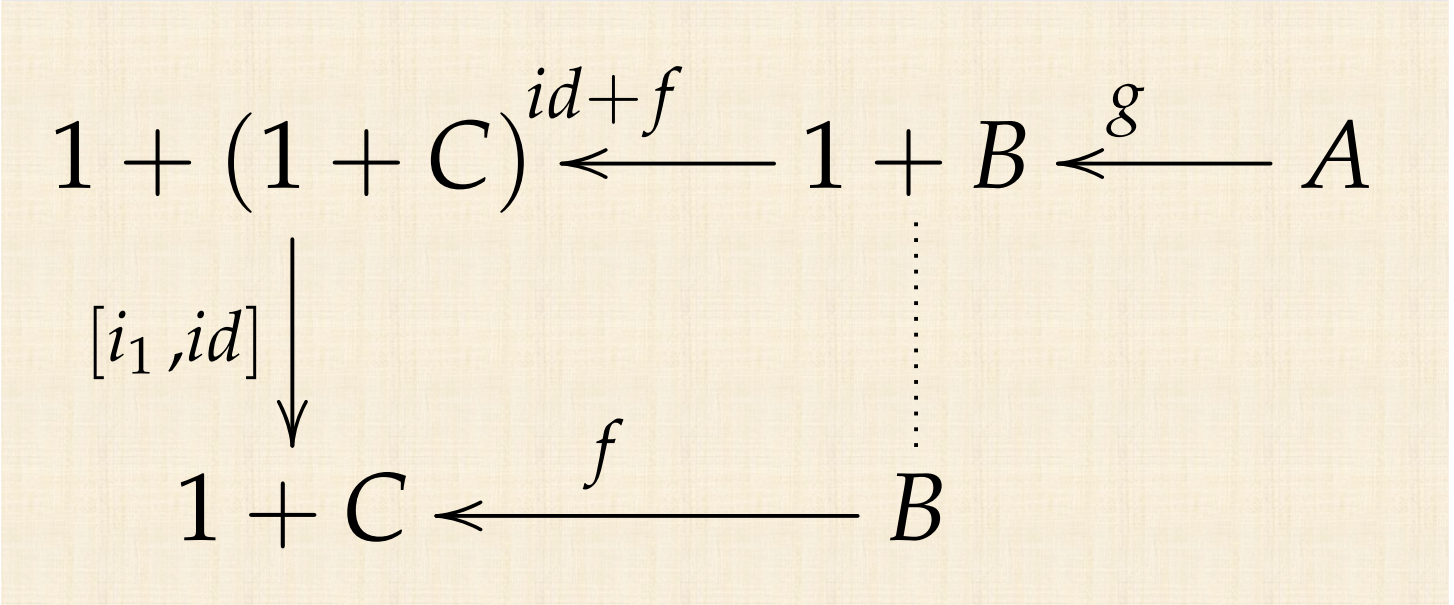
Partial functions



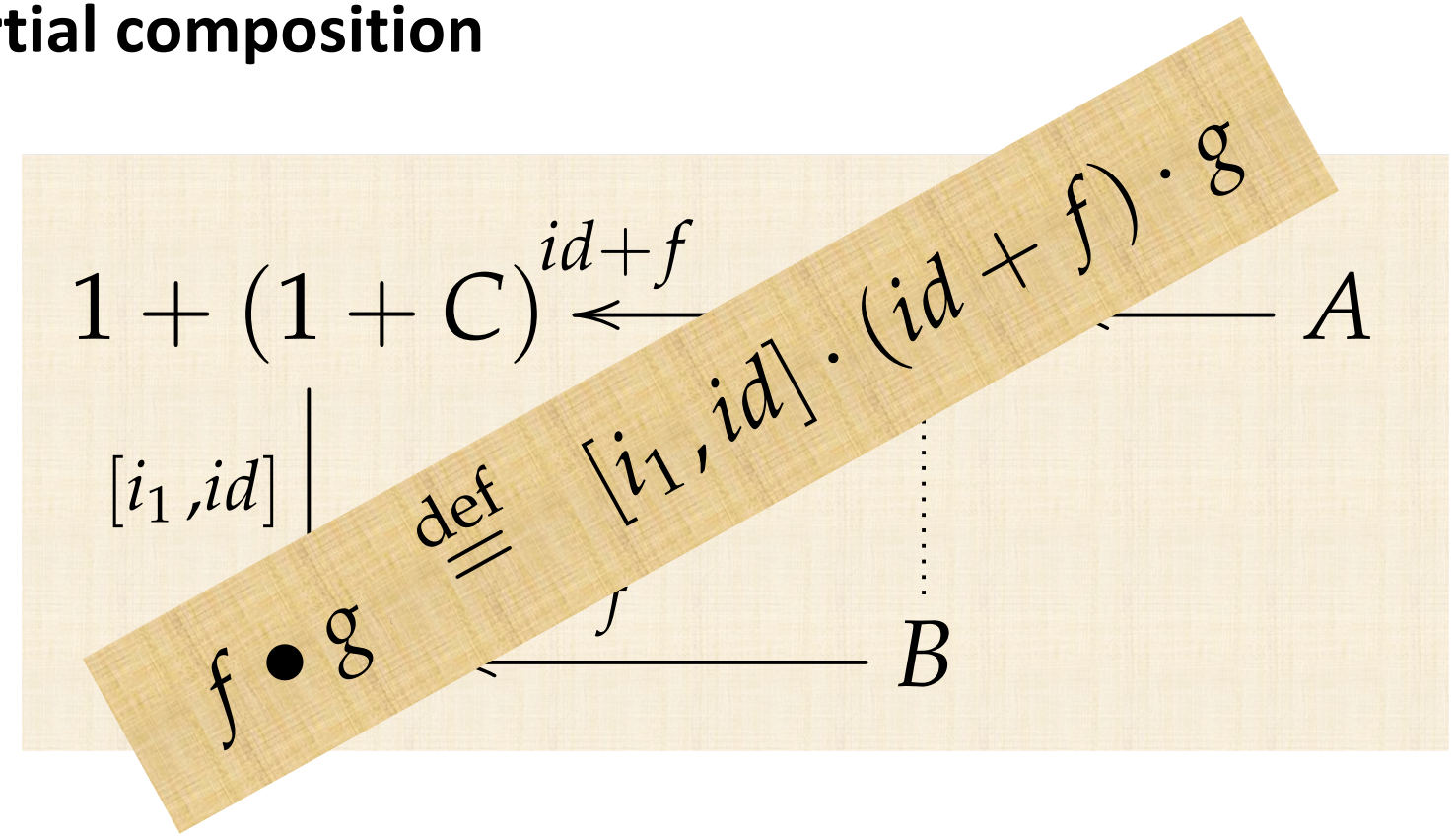
Partial composition



Partial composition

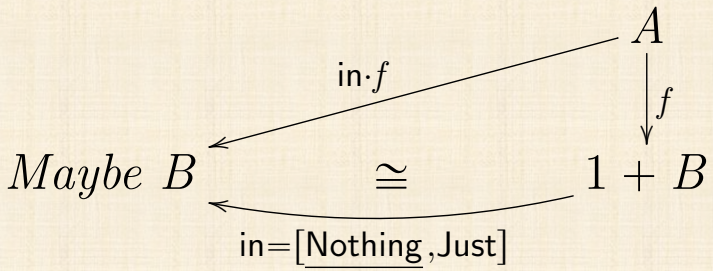


Partial composition



‘Maybe functions’

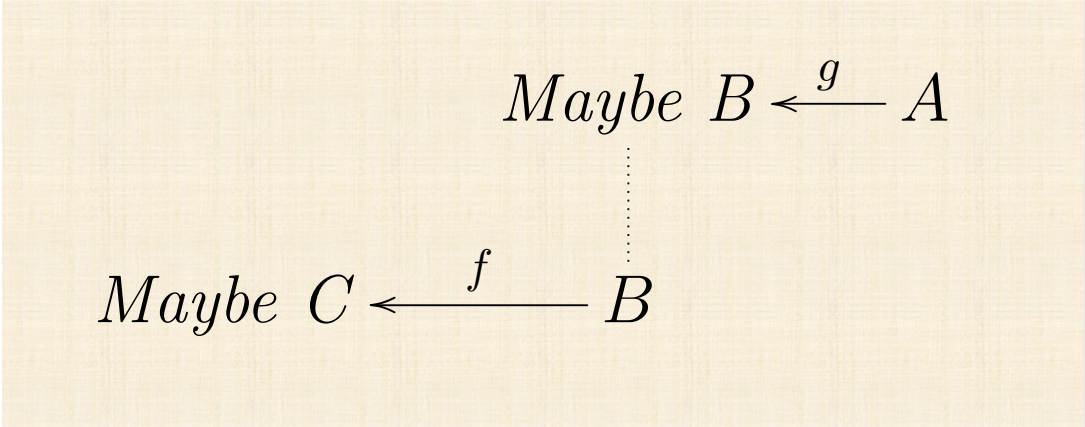
$$Maybe\ B \underset{\text{in}=[\underline{\text{Nothing}},\text{Just}]}{\overset{\text{out}=\text{in}^\circ}{\cong}} 1 + B$$



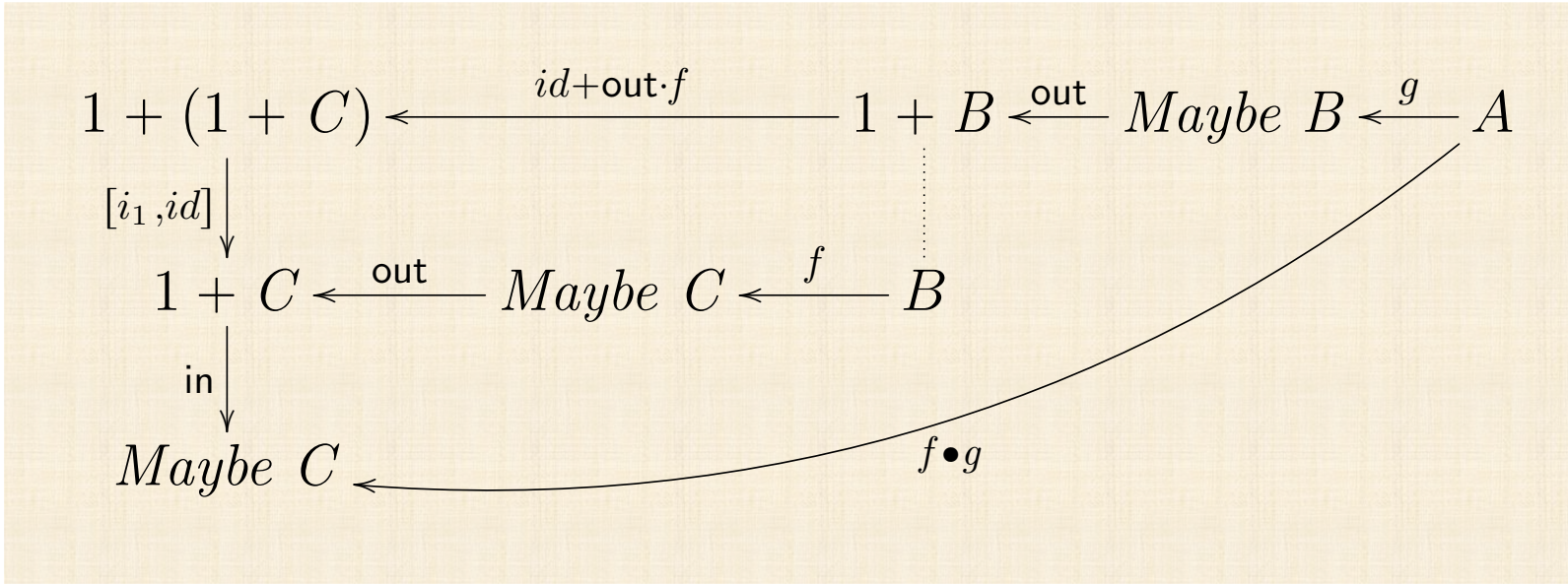
$$\text{out Nothing} = i_1\ ()$$
$$\text{out}\ (\text{Just}\ a) = i_2\ a$$

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

Composing ‘Maybe functions’



Composing ‘Maybe functions’



Composing ‘Maybe functions’

$$
\begin{aligned}
& f \bullet g = \mathsf{in} \cdot [i_1, \mathsf{out} \cdot f] \cdot \mathsf{out} \cdot g \\
\equiv & \quad \{ \text{ fusão-+ e } \mathsf{in} \cdot \mathsf{out} = id \ \} \\
& f \bullet g = [\mathsf{in} \cdot i_1, f] \cdot \mathsf{out} \cdot g \\
\equiv & \quad \{ \text{ introdução da variável } a \ \} \\
& (f \bullet g)\ a = [\mathsf{in} \cdot i_1, f]\ (\mathsf{out}\ (g\ a)) \\
\equiv & \quad \{ \text{ definição de out} \} \\
& (f \bullet g)\ a = \mathbf{if}\ g\ a = \mathsf{Nothing}\ \mathbf{then}\ [\mathsf{in} \cdot i_1, f]\ (i_1\ ())\ \mathbf{else}\ [\mathsf{in} \cdot i_1, f]\ (i_2\ (g\ a)) \\
\equiv & \quad \{ \text{ cancelamento-+ e simplificação} \} \\
& (f \bullet g)\ a = \mathbf{if}\ g\ a = \mathsf{Nothing}\ \mathbf{then}\ \mathsf{Nothing}\ \mathbf{else}\ f\ (g\ a)
\end{aligned}
$$

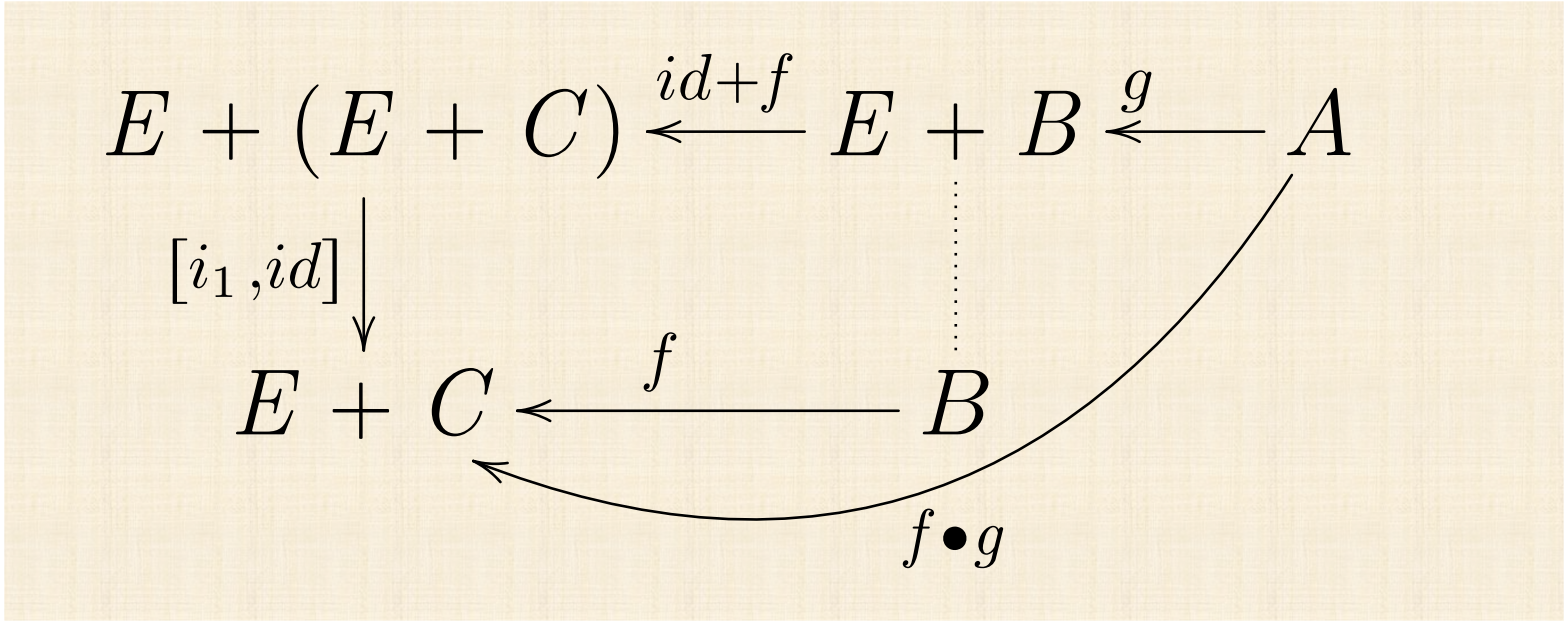
Error messages

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

$$B \xleftarrow{f} A$$

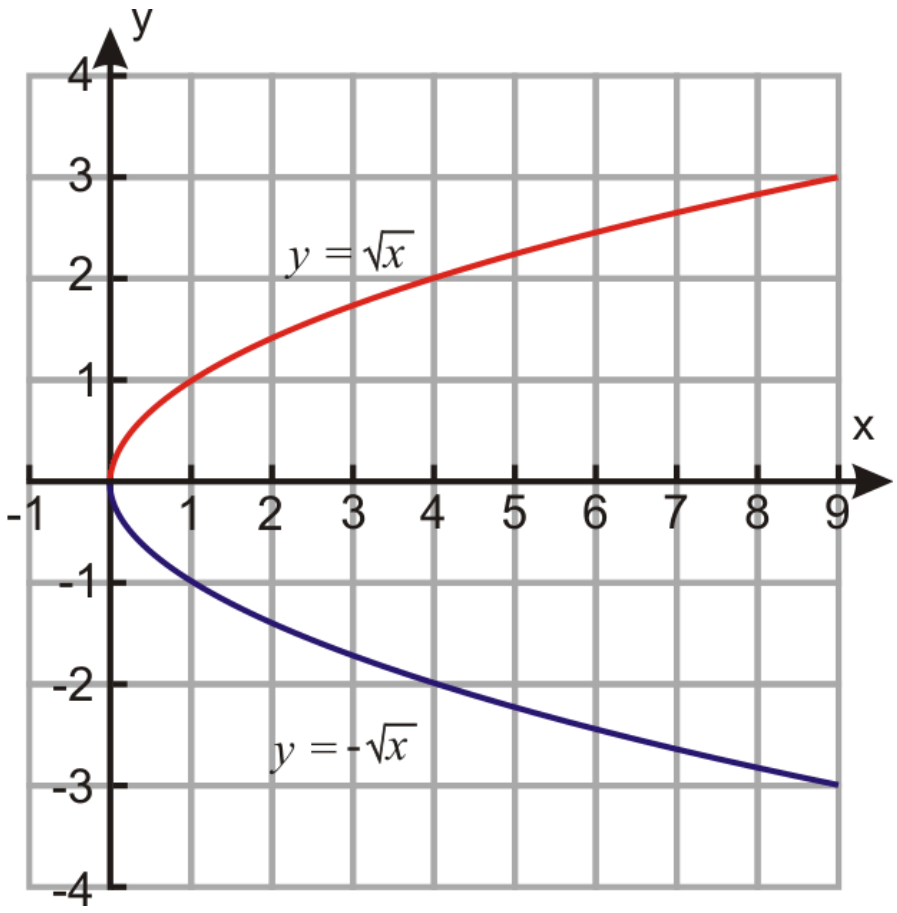
$$E + B \xleftarrow{i_2 \cdot f} A$$

Handling error messages




Unchecked Exception
Checked Exception
Error
Exception Handling

Square root “function”

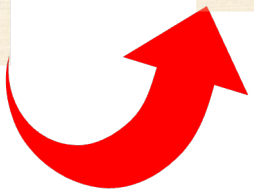


'Undecided' functions

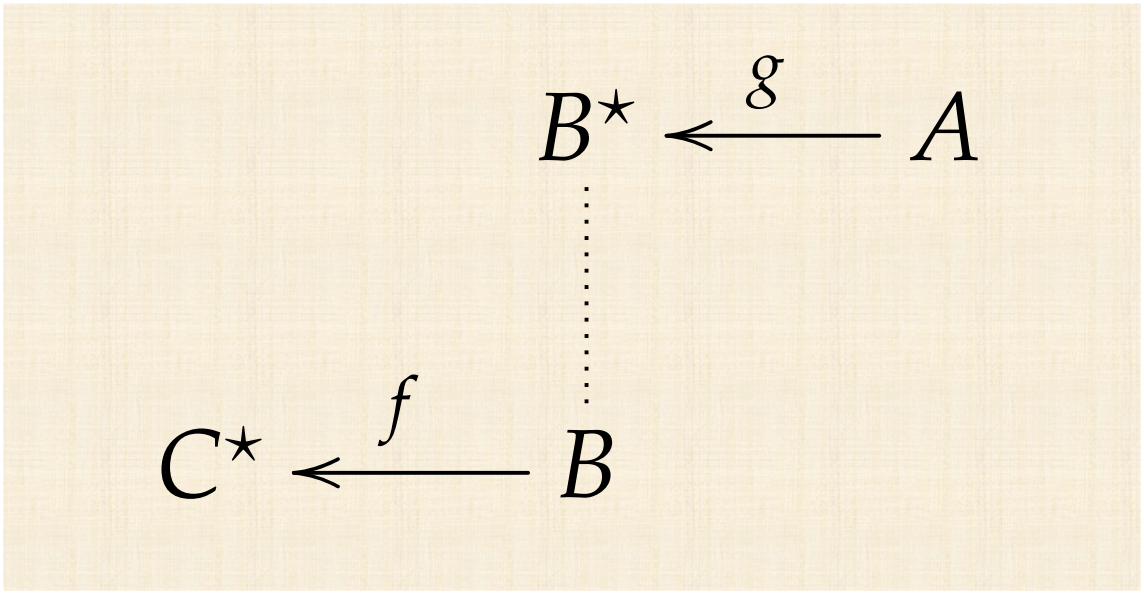
$$B^{\star} \xleftarrow{g} A$$

$$B \xleftarrow{f} A$$

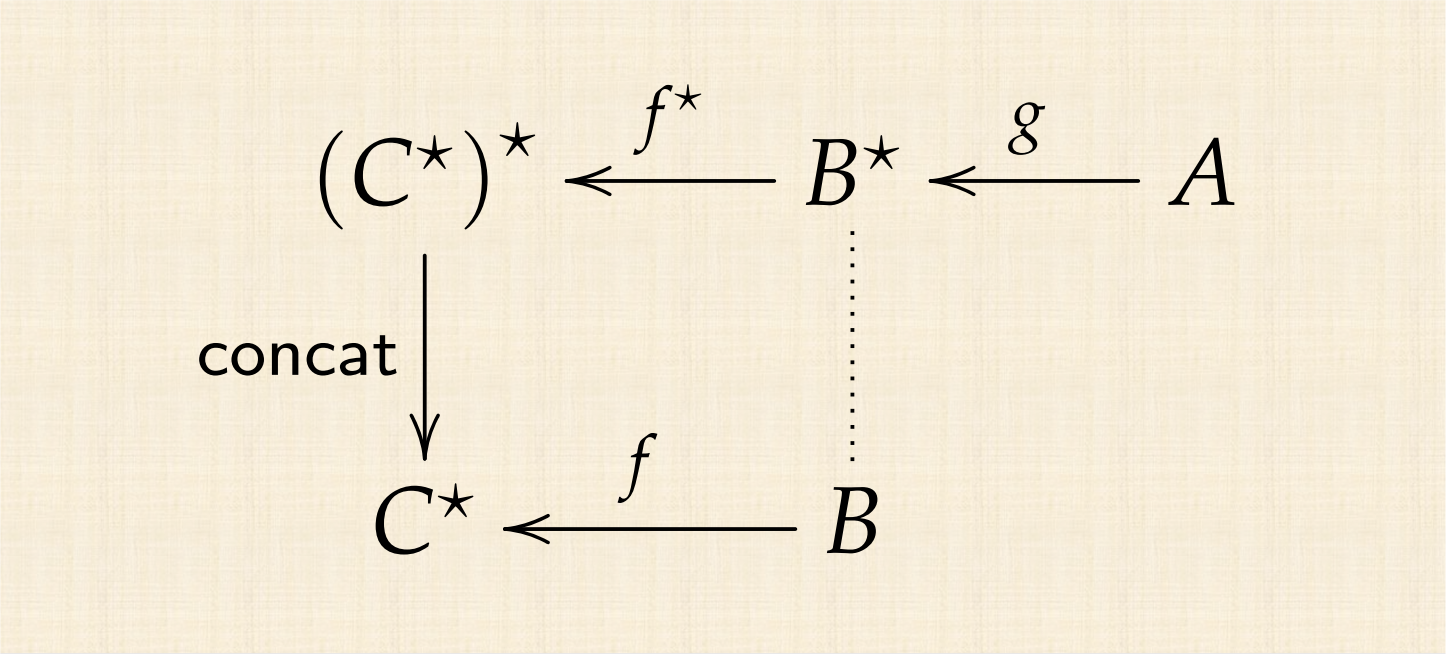
$$B^{*} \xleftarrow{\mathsf{singl}\cdot f} A$$



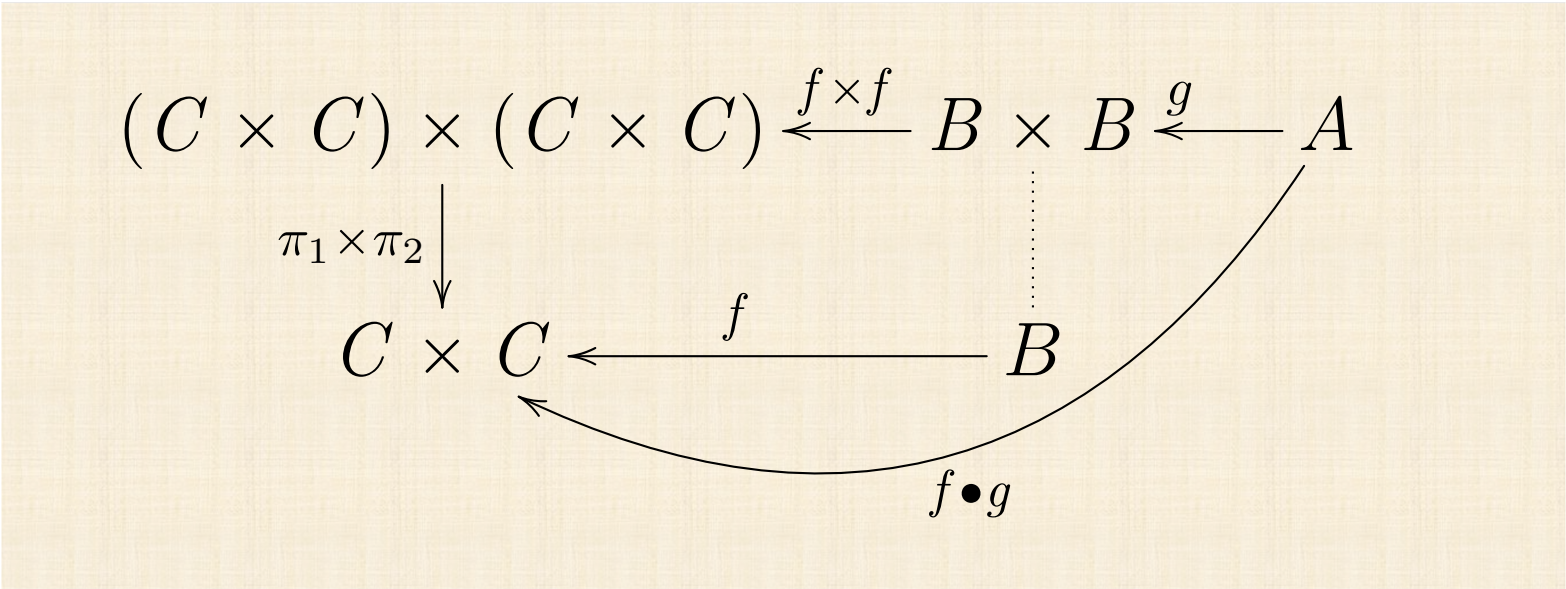
Composing 'undecided' functions



Composing 'undecided' functions



Composing functions that yield pairs



FUNCTORS so far

$$\mathbf{T}\ X = 1 + X$$
$$\mathbf{T}\ X = Maybe\ X$$
$$\mathbf{T}\ X = E + X$$
$$\mathbf{T}\ X = X^*$$
$$\mathbf{T}\ X = X \times X$$

Similar structure

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

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

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

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

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

MONAD

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

MONAD

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

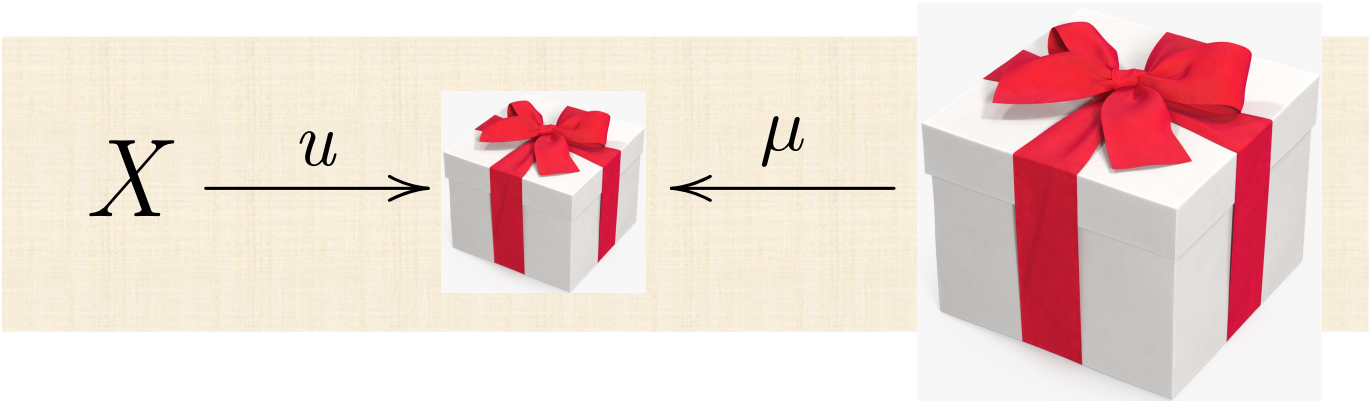
Unit Multiplication

MONAD

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

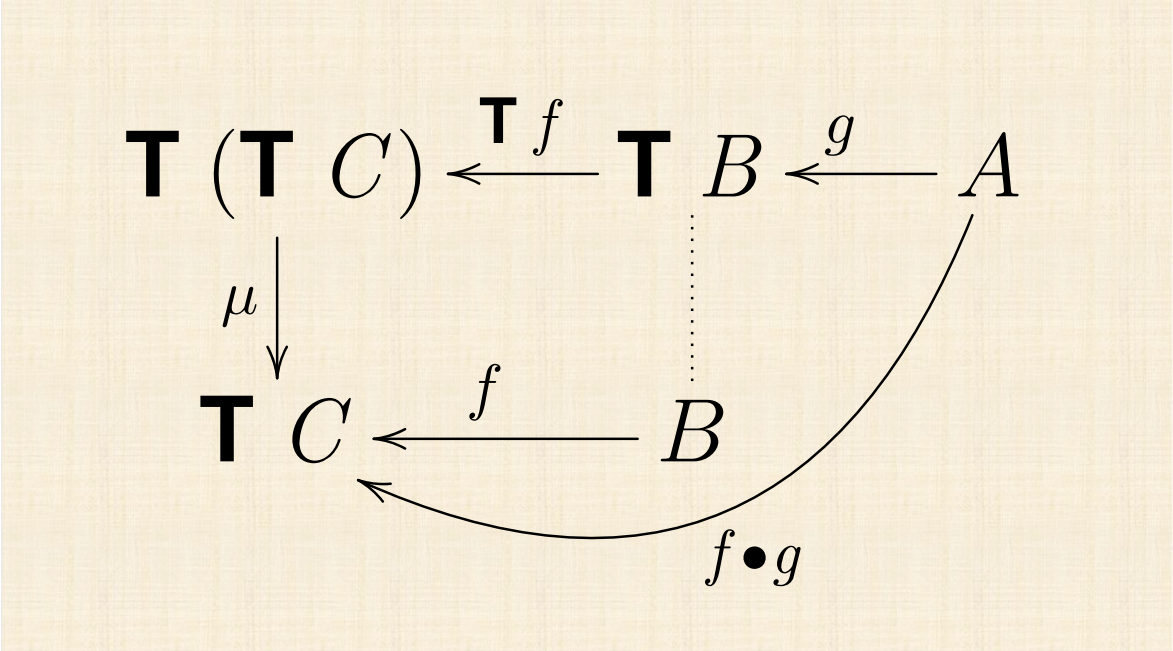
Monad = **Functor** + unit + multiplication

MONAD



Monad = **Functor** + unit + multiplication

Monadic composition



Heinrich Kleisli

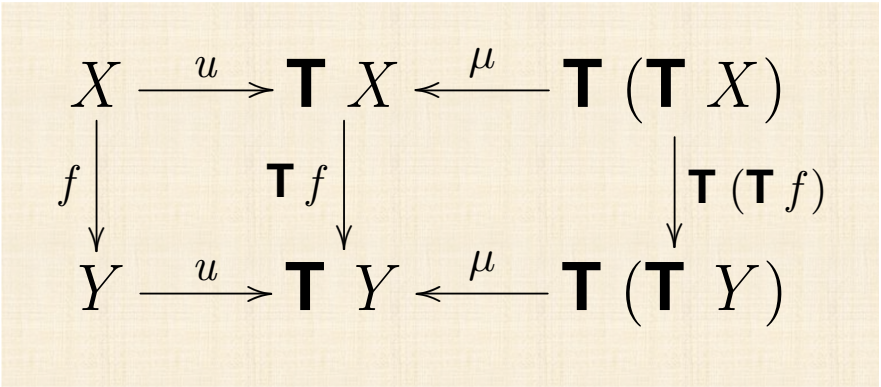
From Wikipedia, the free encyclopedia

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 – natural properties



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

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

Cálculo de Programas
Aula T11