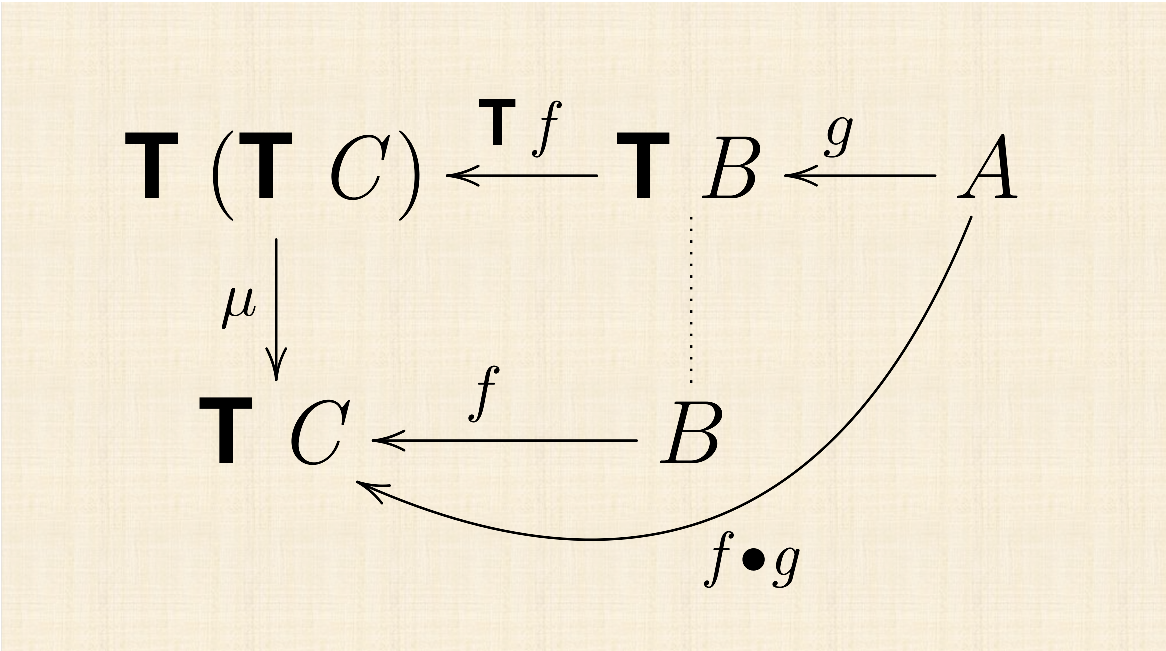
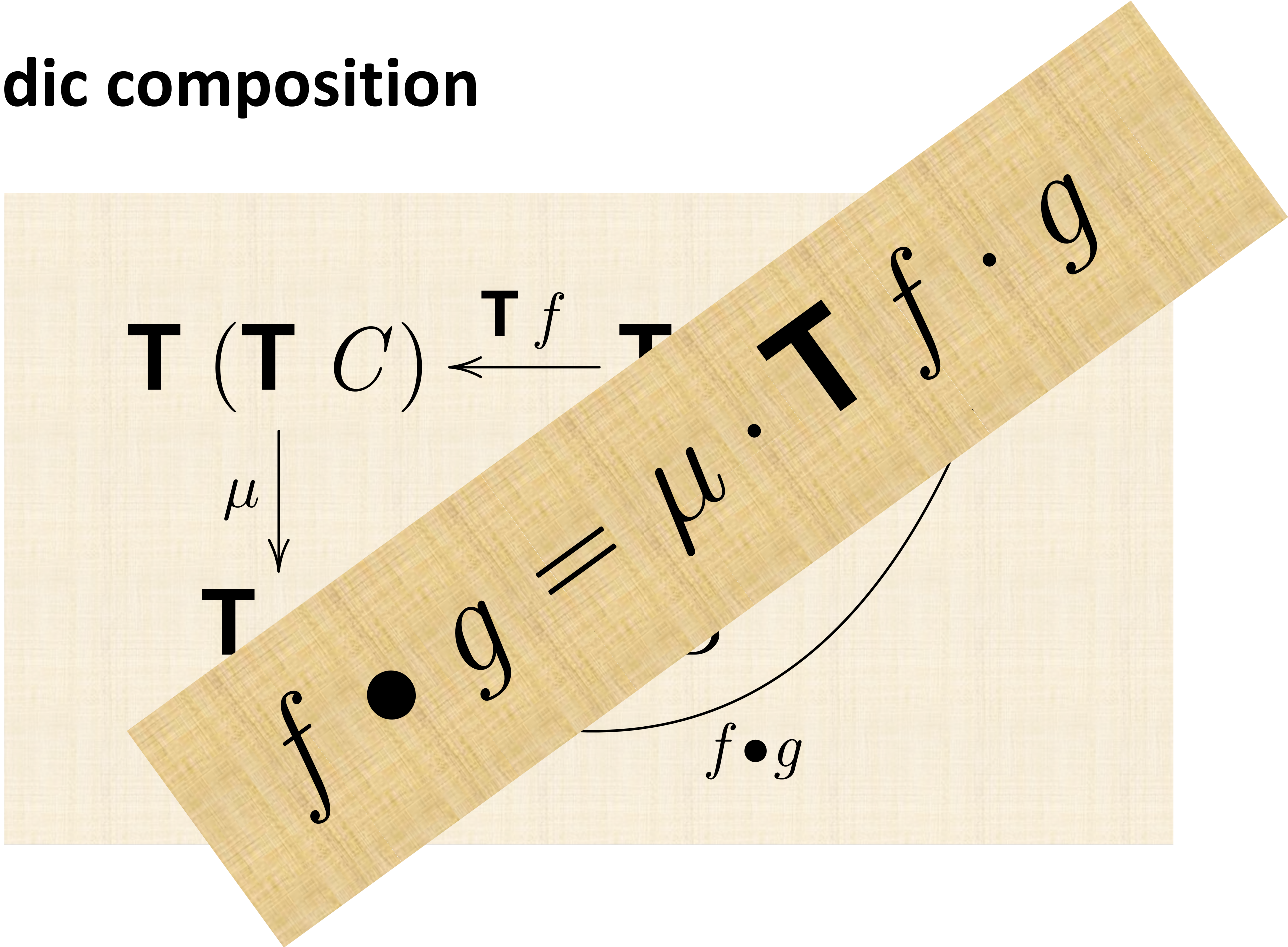
Cálculo de Programas
Aula T11

Monadic composition



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

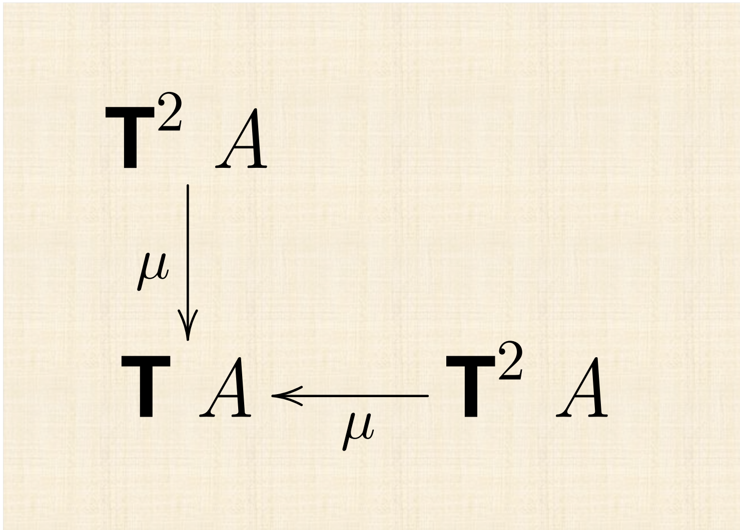
Kleisli earned his Ph.D. at ETH Zurich in 1960, having been supervised by Beno Eckmann and Ernst Specker. His dissertation was on homotopy and abelian categories. He served as an associate professor at the University of Ottawa before relocating to the University of Fribourg in 1966. He became a full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

Monad – natural properties

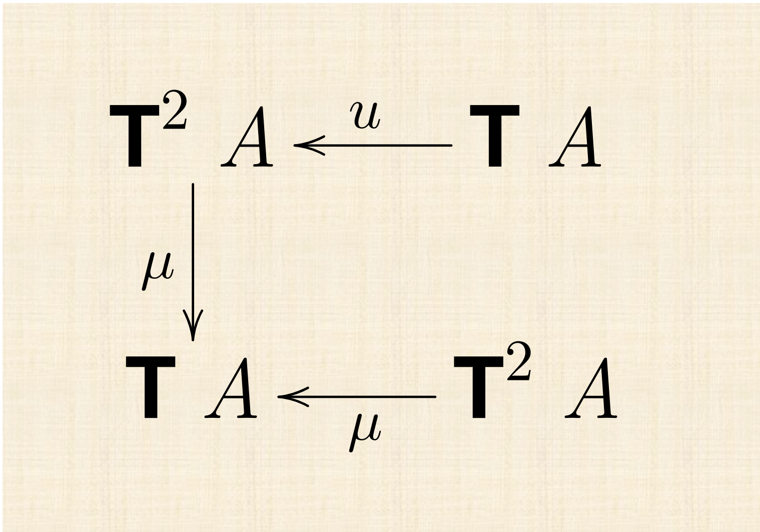
$$
\begin{array}{ccccc}
X & \xrightarrow{u} & \mathbf{T}\,X & \xleftarrow{\mu} & \mathbf{T}\,(\mathbf{T}\,X) \\
\downarrow{\scriptstyle f} & & \downarrow{\scriptstyle \mathbf{T}\,f} & & \downarrow{\scriptstyle \mathbf{T}\,(\mathbf{T}\,f)} \\
Y & \xrightarrow{u} & \mathbf{T}\,Y & \xleftarrow{\mu} & \mathbf{T}\,(\mathbf{T}\,Y)
\end{array}
$$

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

Monad – multiplication…



Monad – multiplication *versus* unit



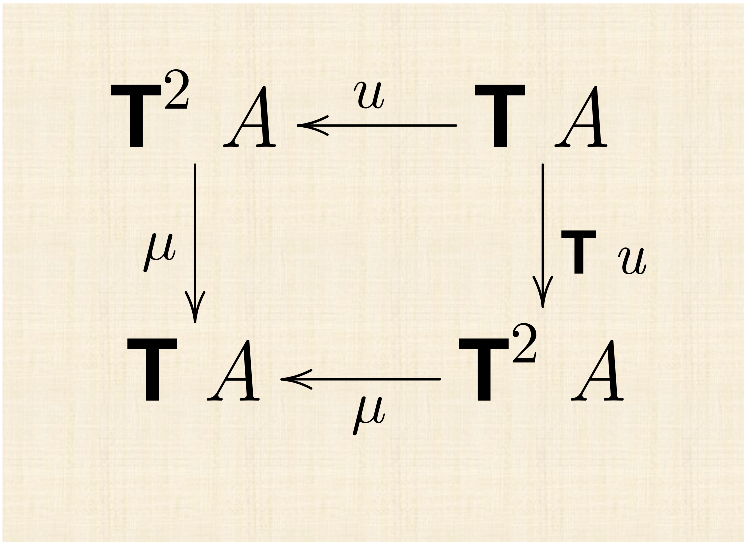
Monad – multiplication *versus* unit

$\mathbf{T}\ X \xleftarrow{u} X$

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

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

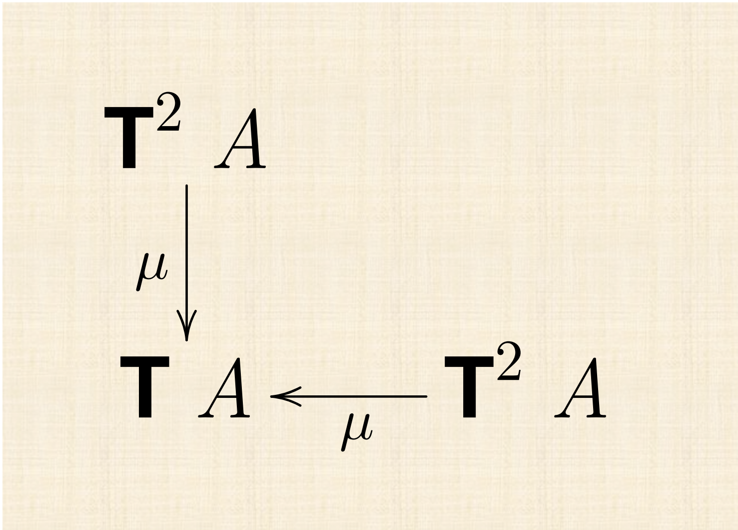
Monad – multiplication *versus* unit



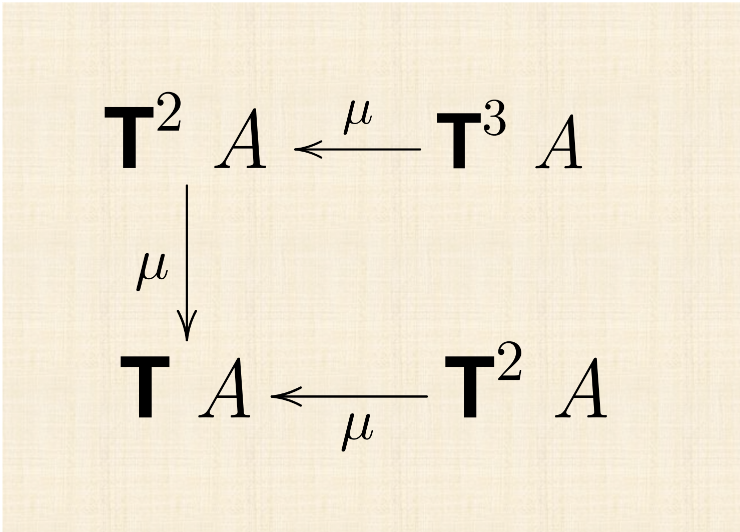
Monad – multiplication *versus* unit

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

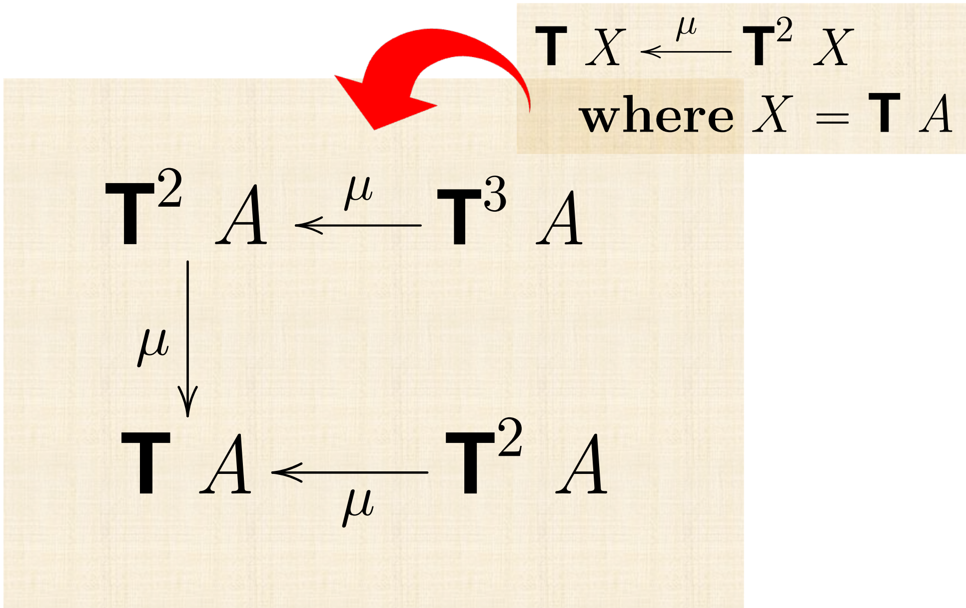
Monad – multiplication *versus* multiplication



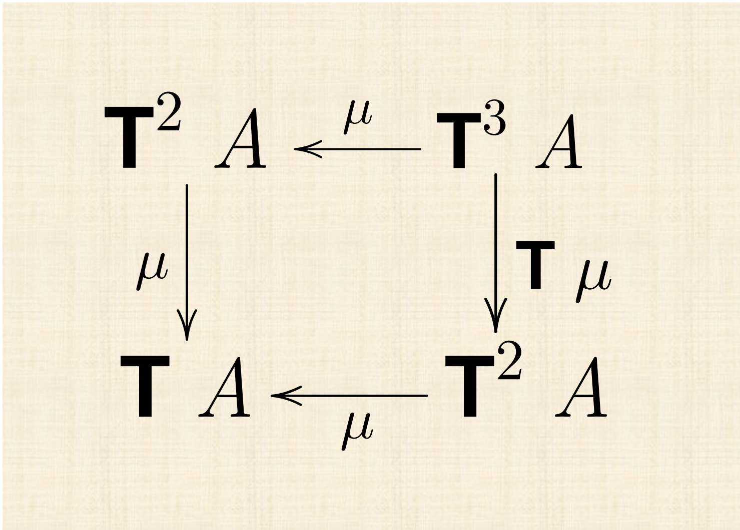
Monad – multiplication *versus* multiplication



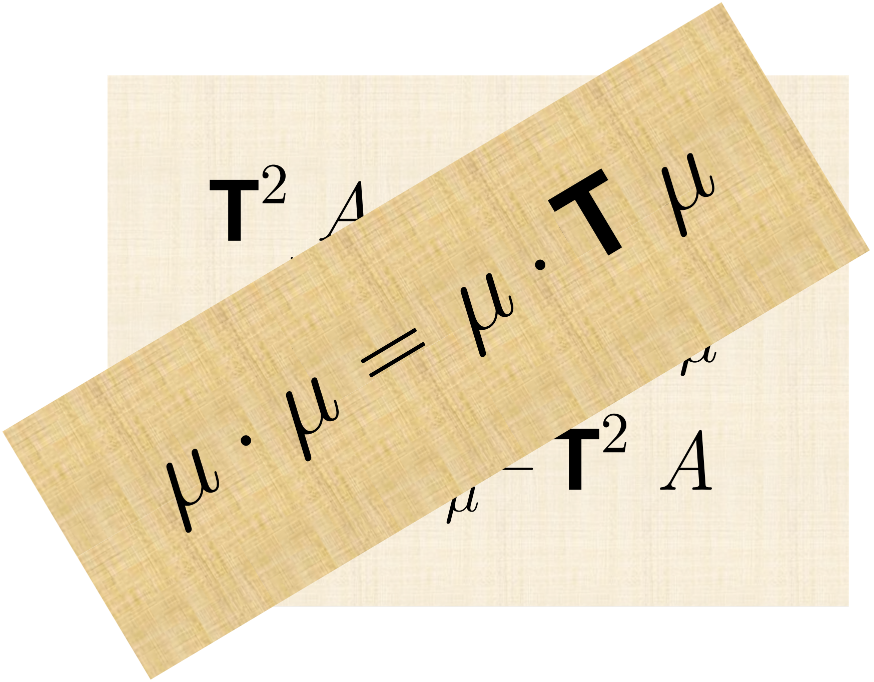
Monad – multiplication *versus* multiplication



Monad – multiplication *versus* multiplication



Monad – multiplication *versus* multiplication



Monad – laws of unit and multiplication

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

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

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

Monad – unit of composition

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

$\equiv$ { definition of $f \bullet g$, twice }

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

$\equiv$ { natural-u and $\mu \cdot \mathbf{T}\, u = id$ }

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

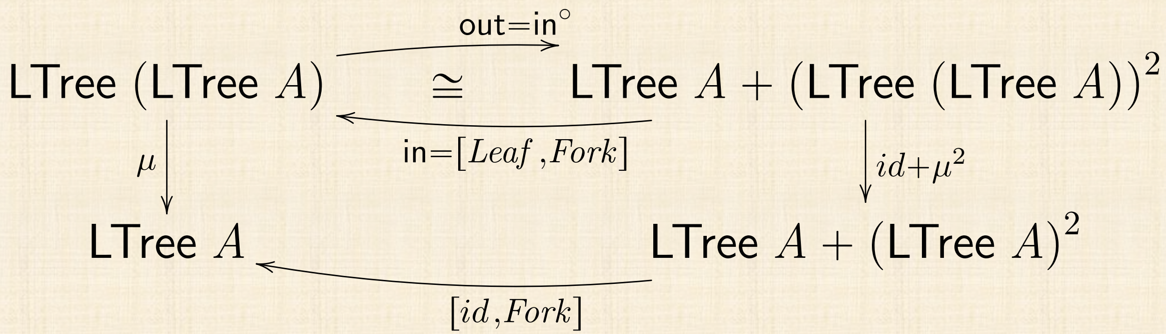
$\equiv$ { $\mu \cdot u = id$ }

$$f = f$$

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

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

LTree monad



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

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

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

LTree monad

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

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

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

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

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

LTree monad

$$\mu \cdot \mathsf{LTree}\ Leaf = id$$

$\equiv$ { definition of μ }

$$(\![[id, Fork]]\!) \cdot \mathsf{LTree}\ Leaf = id$$

$\equiv$ { cata-absorption }

$$(\![[id, Fork] \cdot (Leaf + id)]\!) = id$$

$\equiv$ { +-absorption }

$$(\![[Leaf, Fork]]\!) = id$$

$\equiv$ { cata-reflection }

$$true$$

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

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

```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"

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

$$f \bullet id = ?$$

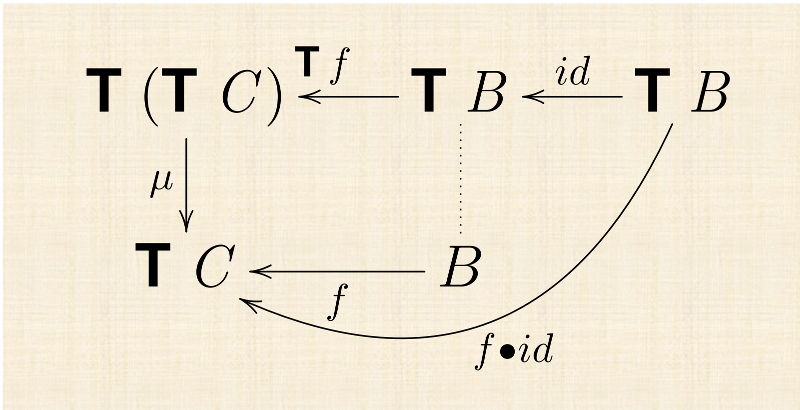
$$f \bullet u = f$$

Monad "binding"

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

$$f \bullet id = \,?$$

$$f \bullet u = f$$

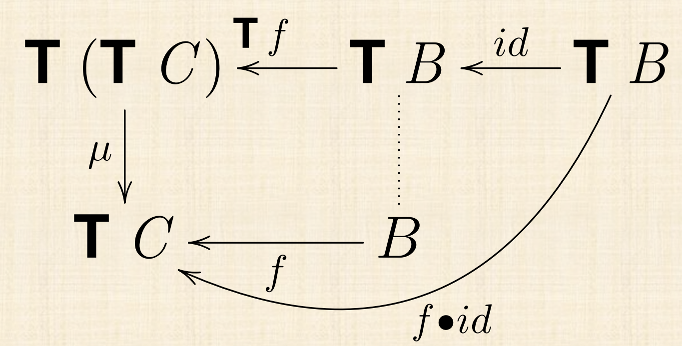


Monad “binding”

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

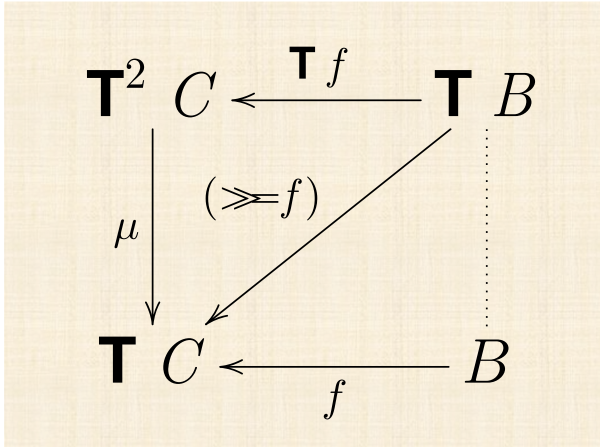
$$f \bullet id = \mu \cdot \mathbf{T}\, f$$

$$f \bullet u = f$$



Monad "binding"

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

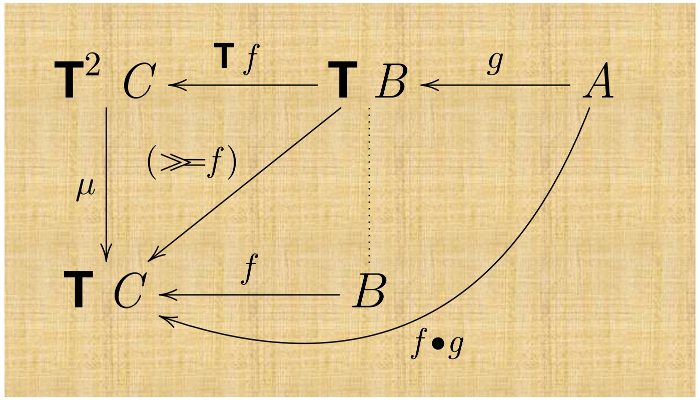


$$(\gg\!= f) = f \bullet id$$

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

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)

```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

$$\text{return} = u$$

Identity monad

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

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

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

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

Monads – summary

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

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

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

Formulae sheet

MÓNADAS

Multiplicação	$\mu \cdot \mu = \mu \cdot \mathsf{T}\,\mu$	(62)
Unidade	$\mu \cdot u = \mu \cdot \mathsf{T}\,u = id$	(63)
Natural-u	$u \cdot f \quad = \quad \mathsf{T}\,f \cdot u$	(64)
Natural-μ	$\mu \cdot \mathsf{T}\,(\mathsf{T}\,f) \quad = \quad \mathsf{T}\,f \cdot \mu$	(65)
Composição monádica	$f \bullet g \quad = \quad \mu \cdot \mathsf{T}\,f \cdot g$	(66)
Associatividade-$\bullet$	$f \bullet (g \bullet h) = (f \bullet g) \bullet h$	(67)
Identidade-$\bullet$	$u \bullet f = f = f \bullet u$	(68)
Associatividade-$\bullet/\cdot$	$(f \bullet g) \cdot h \quad = \quad f \bullet (g \cdot h)$	(69)
Associatividade-$\cdot/\bullet$	$(f \cdot g) \bullet h \quad = \quad f \bullet (\mathsf{T}\,g \cdot h)$	(70)
μ **versus** $\bullet$	$id \bullet id \quad = \quad \mu$	(71)

Formulae sheet

Composição monádica	$(f \bullet g)\ a = \mathbf{do}\ \{b \leftarrow g\ a; f\ b\}$	(86)
'Binding as μ'	$x \ggg f \quad = \quad (\mu \cdot \mathsf{T}\ f) x$	(87)
Notação-do	$\mathbf{do}\ \{x \leftarrow a; b\} \quad = \quad a \ggg (\lambda x \rightarrow b)$	(88)
'μ as binding'	$\mu\ x \quad = \quad x \ggg id$	(89)
Sequenciação	$x \gg y \quad = \quad x \ggg \underline{y}$	(90)

Monad =
“racing”
functor

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

MONADS: do-notation

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

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

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

MONADS: do-notation

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

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

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

MONADS: do-notation

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

$$
\begin{aligned}
& x \gg\!= f \\
= \quad & \{ \text{ definition of } (\gg\!= f) \ \} \\
& (f \bullet id)\ x \\
= \quad & \{ \text{ do-notation } \} \\
& \mathbf{do}\ \{ b \leftarrow x; f\ \ b \}
\end{aligned}
$$

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

MONADS: do-notation

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

$$x \gg\!\!= f$$

$= \{$ definition of $(\gg$

$$(f \bullet id)\ x$$

$= \{$

$\mathbf{do}\ \{$ … $\}$

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

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

MONADS: do-notation

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

$$x \gg\!= f$$

$=$ { definition of ($\gg$

$$(f \bullet id)\ x$$

$=$ {

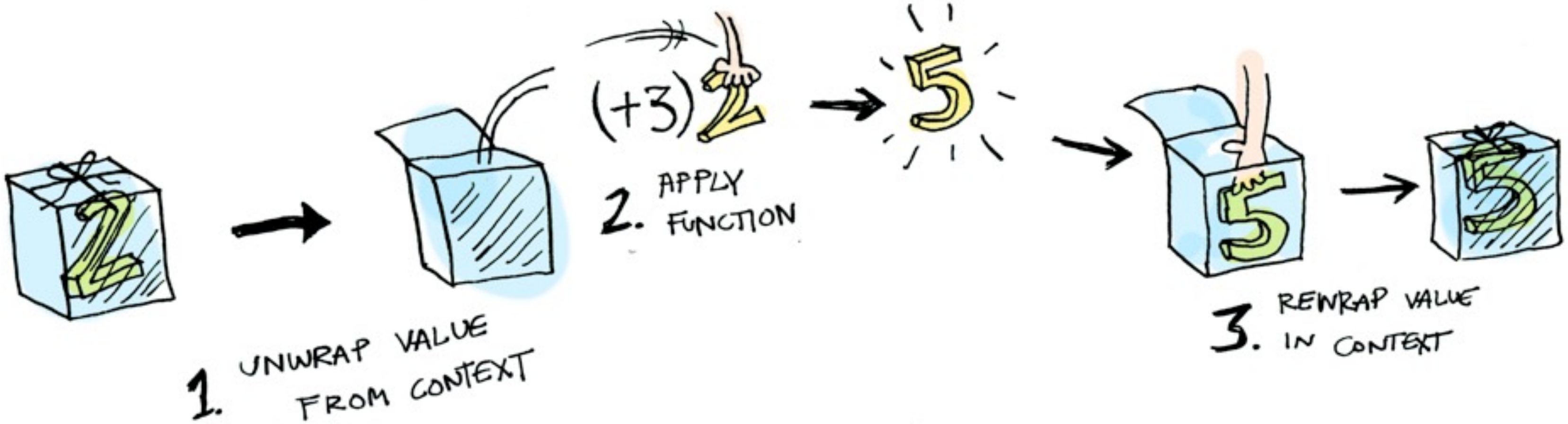
do {

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

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

MONADS: do-notation

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



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

MONADS: do-notation

Recall law

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

Particular case : $g, h := u, id$:

MONADS: do-notation

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

Particular case : $g, h := u, id$:

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

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

$\equiv$ { natural-id; unit u }

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

$\equiv$ { go pointfree }

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

$\equiv$ { introduce d-notation }

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

MONADS: do-notation

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

…icular case : $g, h := u, id$:

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

$\equiv$ { natural-id; unit u }

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

$\equiv$ { go pointfree }

$((u \cdot f) \bullet id)$ …

$\equiv$ { i… …n }

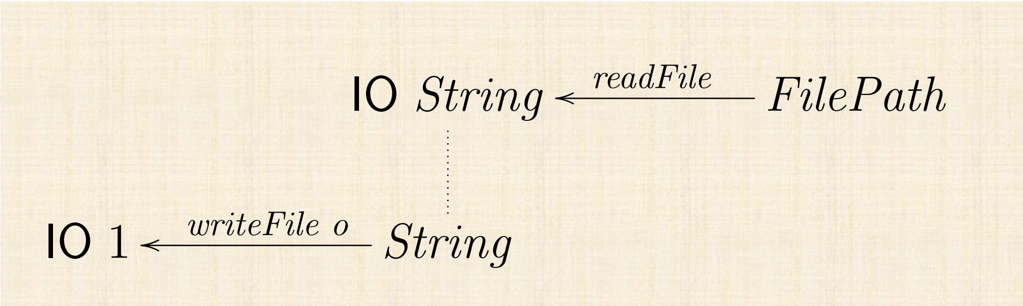
$\mathsf{T}\ f\ x$ … $\leftarrow x;$ return $(f\ b)\}$

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

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

I/O monad : interfacing with the file system

I/O monad : interfacing with the file system



$$
\begin{array}{ll}
 & copy\ i\ o = (writeFile\ o \bullet readFile)\ i \\
\equiv & \quad \{ \text{ do-notation } \} \\
 & copy\ i\ o = \mathbf{do}\ \{ s \leftarrow readFile\ i;\ writeFile\ o\ s \}
\end{array}
$$

O monad (trivial) da identidade.

MONADS: laws written using do-notation

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

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

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

$\equiv$ { go pointwise }

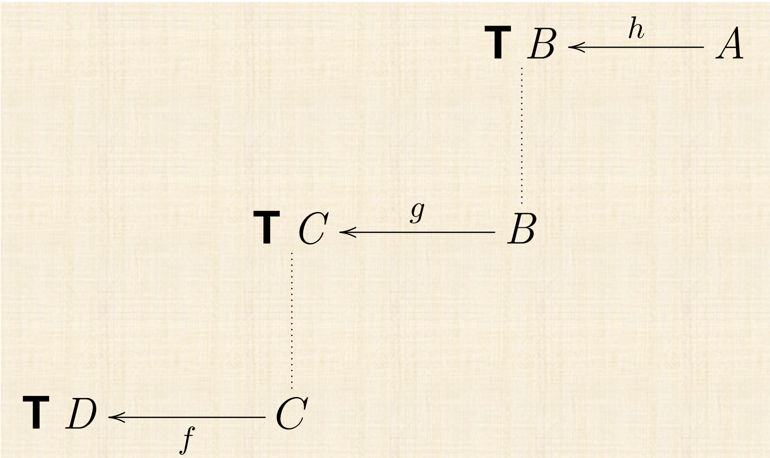
$$(u \bullet f)\ a = f\ a = (f \bullet u)\ a$$

$\equiv$ { **do**-notation }

$$\mathbf{do}\ \{ b \leftarrow f\ a; \text{return}\ b \} = f\ a = \mathbf{do}\ \{ b \leftarrow \text{return}\ a; f\ b \}$$

MONADS: laws written using do-notation

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

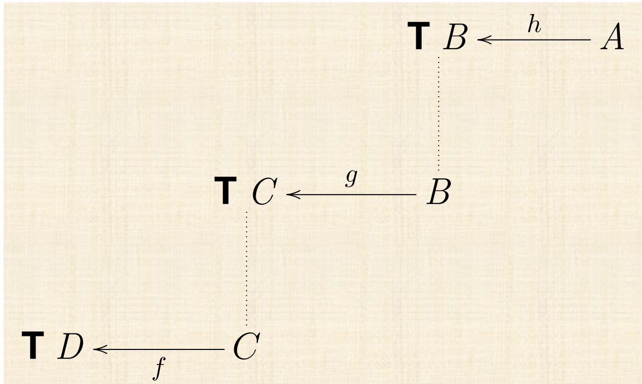


$$(f \bullet g)\ a = \mathbf{do}\ \{ b \leftarrow g\ a; f\ b \}$$

MONADS: laws written using do-notation

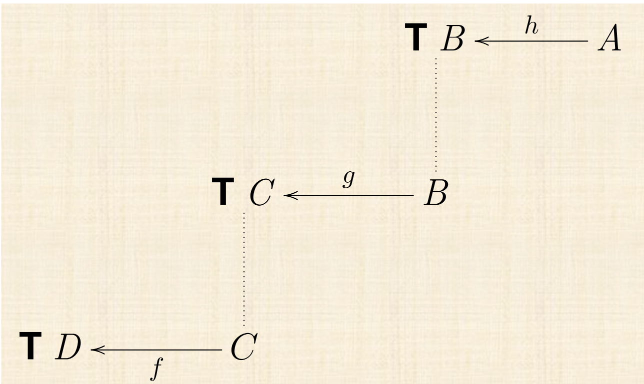
$$f \bullet (g \bullet h) = (f \bullet g) \bullet h$$

$$
\begin{aligned}
& (f \bullet (g \bullet h))\ a \\
\equiv\quad & \{\ \text{do-notation}\ \} \qquad (f \bullet g)\ a = \mathbf{do}\ \{b \leftarrow g\ a; f\ b\} \\
& \mathbf{do}\ \{c \leftarrow (g \bullet h)\ a; f\ c\} \\
\equiv\quad & \{\ \text{do-notation}\ \} \qquad (f \bullet g)\ a = \mathbf{do}\ \{b \leftarrow g\ a; f\ b\} \\
& \mathbf{do}\ \{c \leftarrow \mathbf{do}\ \{b \leftarrow h\ a; g\ b\}; f\ c\}
\end{aligned}
$$



MONADS: laws written using do-notation

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



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

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

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

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

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

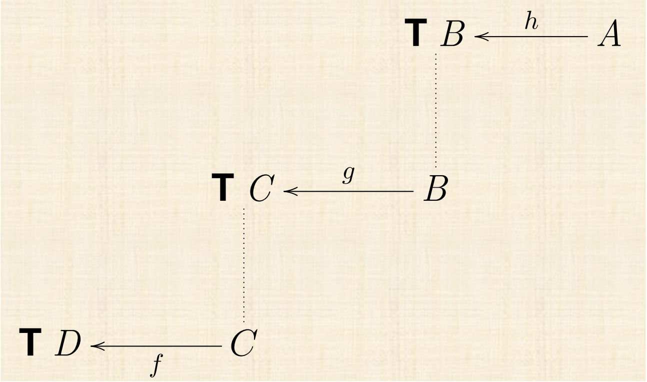
$\equiv \quad \{\ \text{simplify}\ \}$

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

MONADS: laws written using do-notation

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

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



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

MONADIC RECURSION: the pointwise way

Code "monadification" made easy

J.N. Oliveira

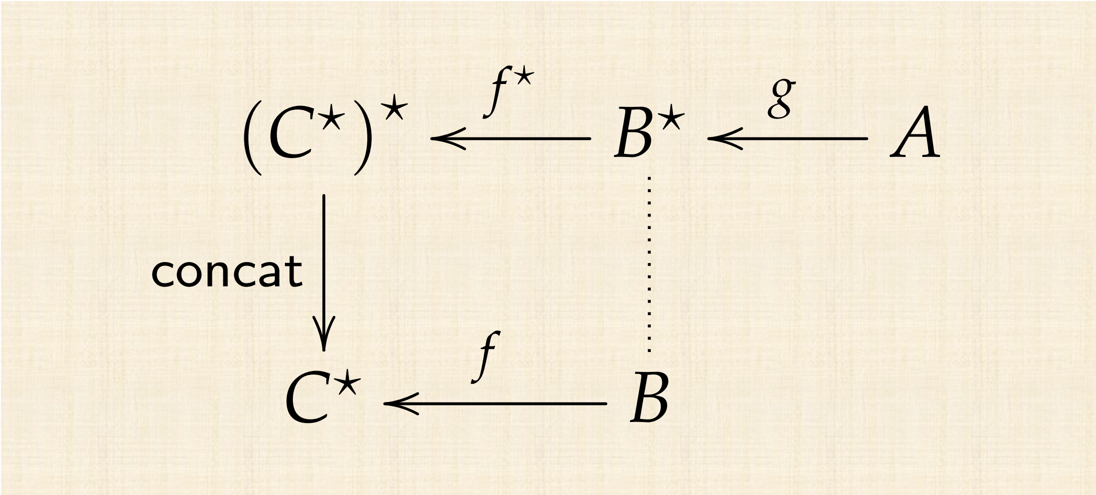
Notes for the MiEI/LCC degrees

University of Minho/DI, June 2010

(last update: May 2020)

MONADS: list compreensions

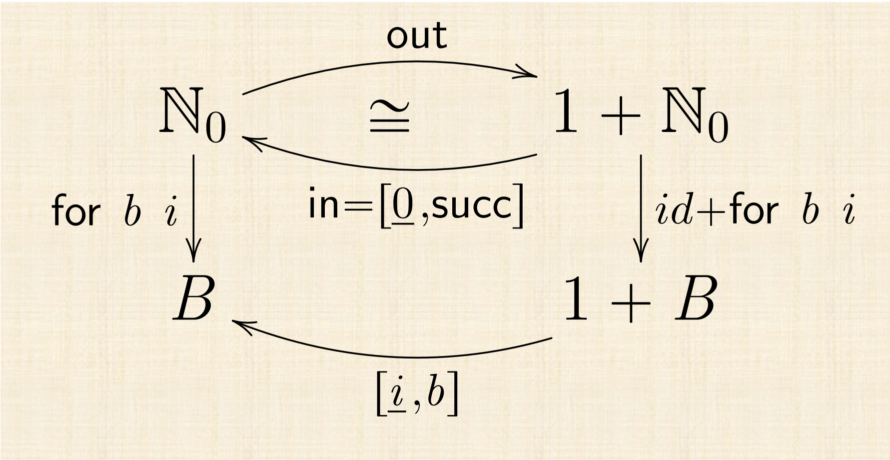
$$[\, e \mid a_1 \leftarrow x_1, \ldots, a_n \leftarrow x_n \,] = \mathbf{do}\ \{\, a_1 \leftarrow x_1; \ldots; a_n \leftarrow x_n; \mathsf{return}\ e \,\}$$



Cálculo de Programas
Aula T12

Monadic recursion

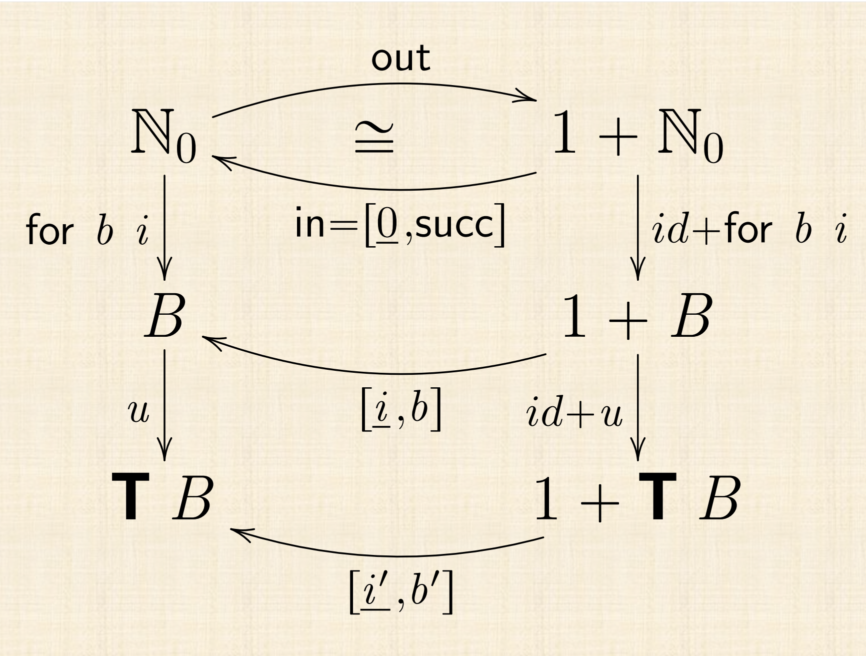
$$X \xrightarrow{u} \mathbf{T}\ X \xleftarrow{\mu} \mathbf{T}\ (\mathbf{T}\ X)$$



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

Monadic **recursion**

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



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

Monadic recursion

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

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

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

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

$\Leftarrow$ { fusão-cata }

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

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

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

Monadic recursion

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

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

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

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

$\Leftarrow \quad \{ \ \text{fusão-cata} \ \}$

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

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

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

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

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

$\Leftarrow \quad \{ \ \text{trivial} \ \}$

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

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

Monadic recursion

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

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

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

$\equiv$ { universal cata }

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

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

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

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

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

Monadic recursion

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

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

Monadic recursion

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

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

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

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

Monadic recursion

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

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

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

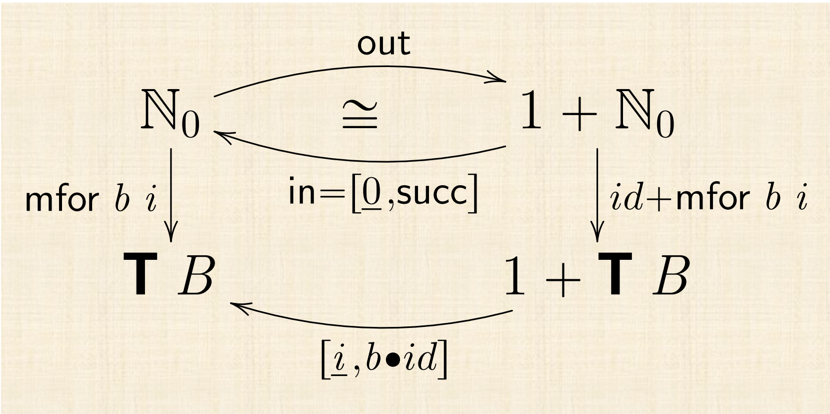
Associatividade-•/· $\quad (f \bullet g) \cdot h \quad = \quad f \bullet (g \cdot h) \qquad (66)$

$$\begin{array}{ll} & (b \bullet (\mathsf{mfor}\ b\ i))\ n \\ = & \quad \{\ id;\ \text{composition}\ ;\ \text{monadic composition}\ \} \\ & (b \bullet id)\ (\mathsf{mfor}\ b\ i\ n) \end{array}$$

Monadic recursion

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

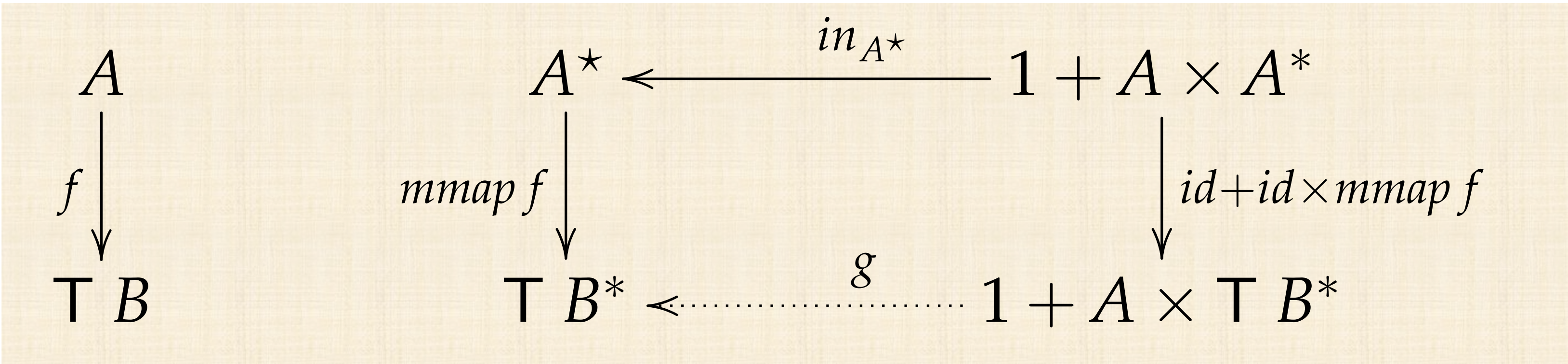
```
mfor b i 0 = i
mfor b i (n+1) = do { x <- mfor b i n ; b x }
```



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

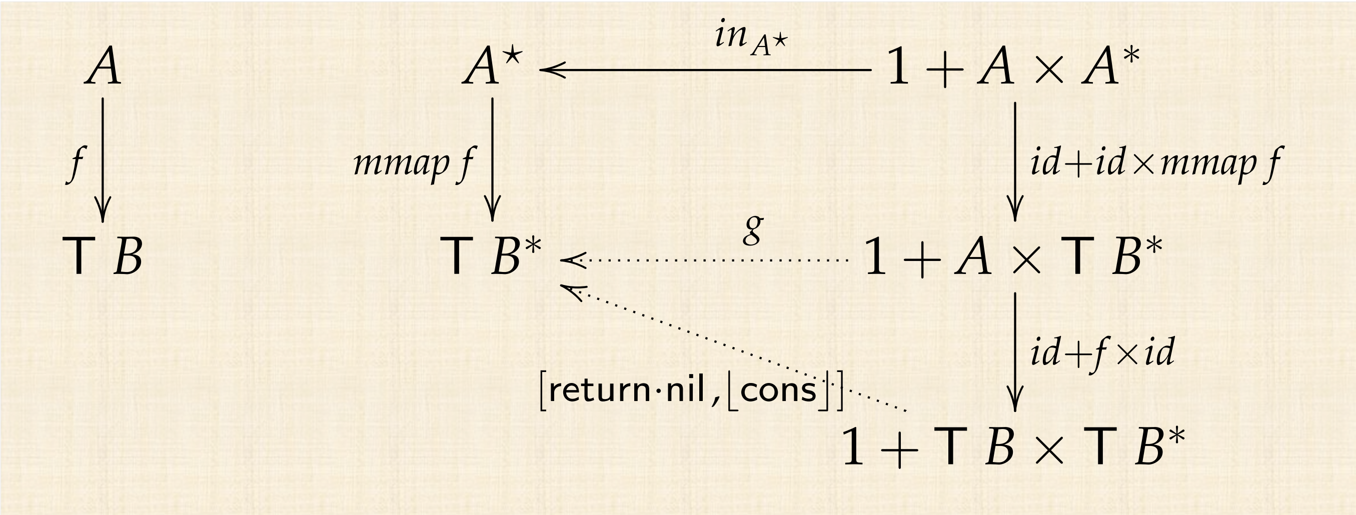
$$\mathsf{mfor}\ b\ i = (\![\ [\underline{i}, b \bullet id]\ ]\!)$$

Monadic map (mmap)

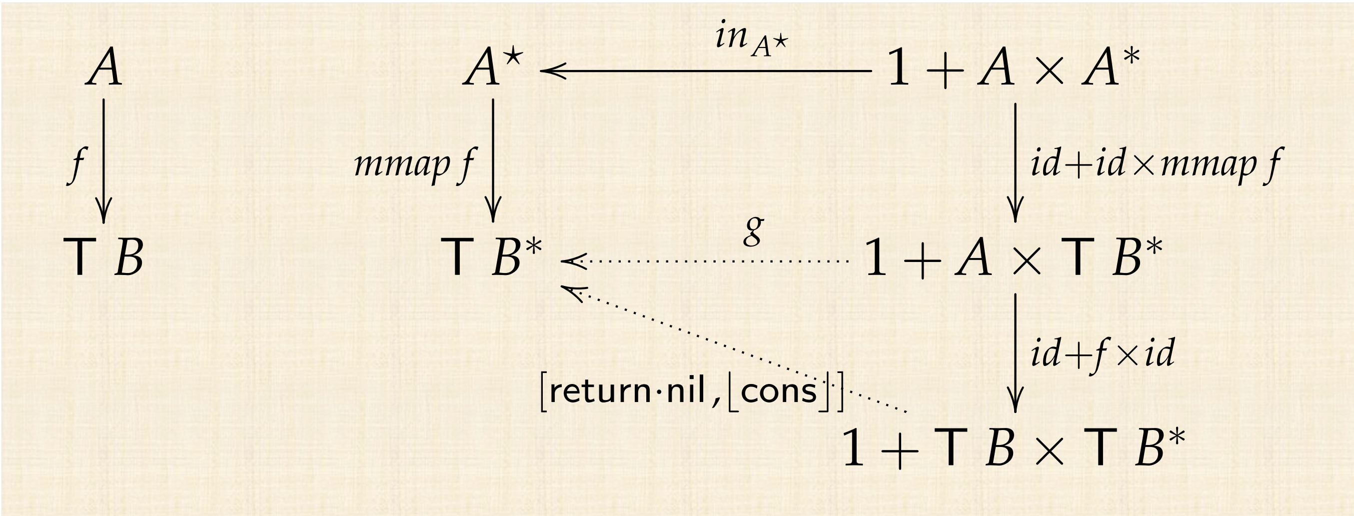


g?

Monadic map (mmap)



Monadic map (mmap)



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

Monadic map (mmap)

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

Monadic map (mmap)

Getting the minimum of a list (if possible…)

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

Get the list of all minima (where possible…)

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

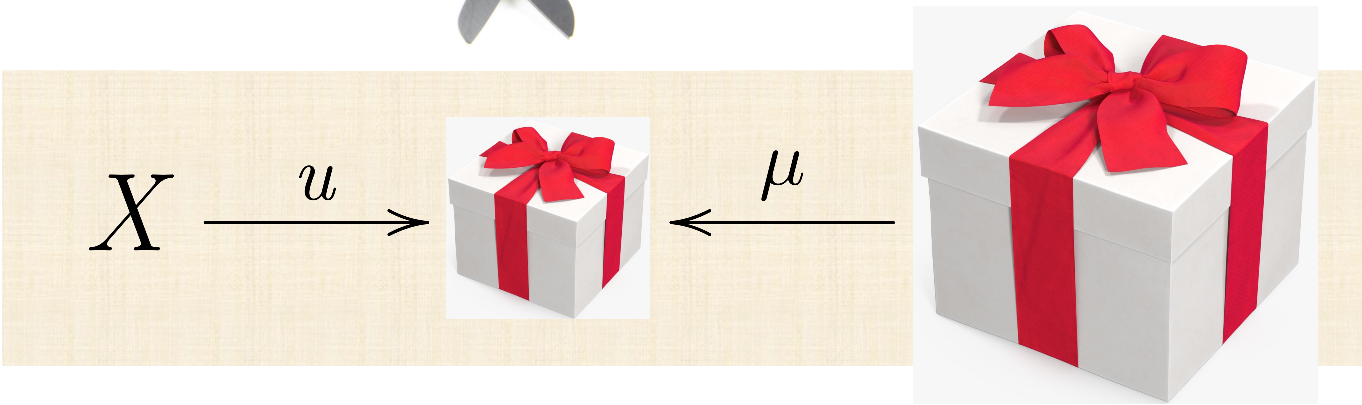
Monadic map (mmap)

Getting the minimum of a list (if possible…)

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

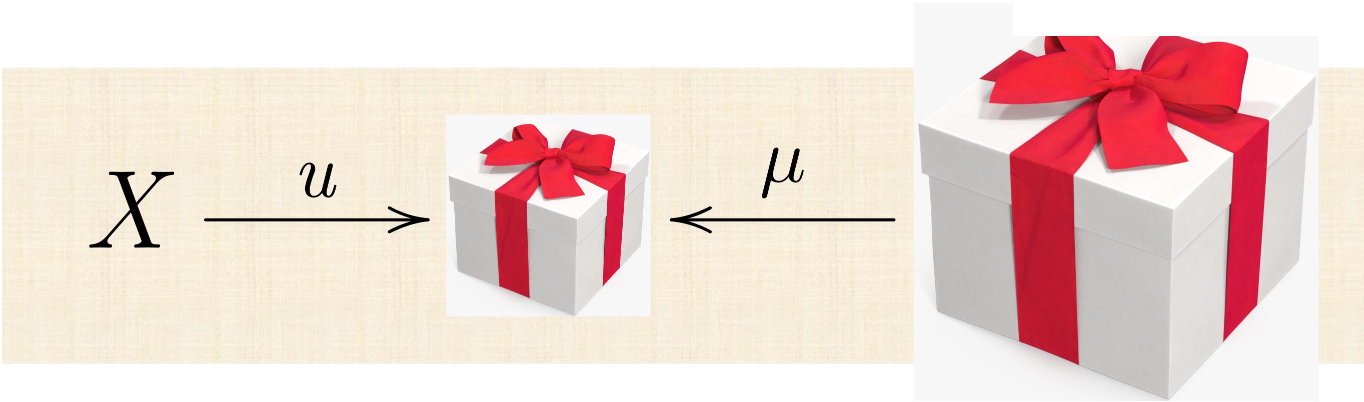
$$
\begin{aligned}
&mgetmin\ [\,] = \mathsf{Nothing}\\
&mgetmin\ [a] = \mathsf{return}\ a\\
&mgetmin\ (h:t) = \mathbf{do}\ \{x \leftarrow mgetmin\ t; \mathsf{return}\ (min\ h\ x)\}
\end{aligned}
$$

MONADS



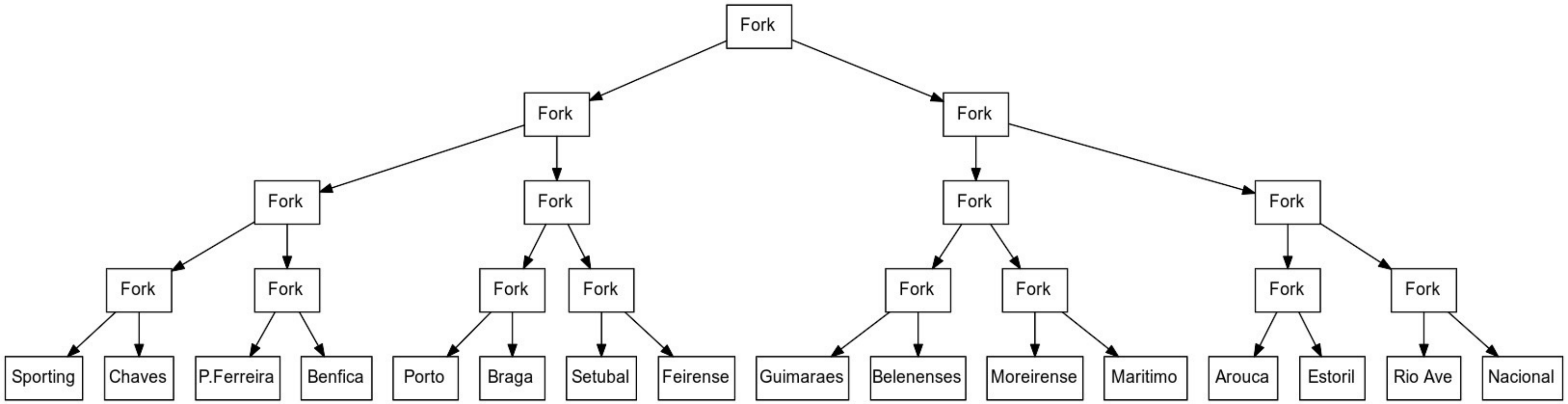

programming
productivity
giants
estimation
potpourri
troubleshooters
hardware
methodologies
manpower
established
compilation
standardized
tackled
prototyping
categorical
project
programmer
staffs
smaller
sigma
shortened
india
techniques
waned
tools
interval
waxed
timelines
iterative
management
promulgated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

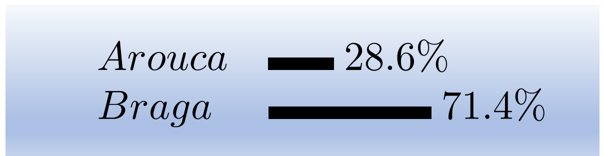
MONADS (recall)



Probabilistic
monadic
programming

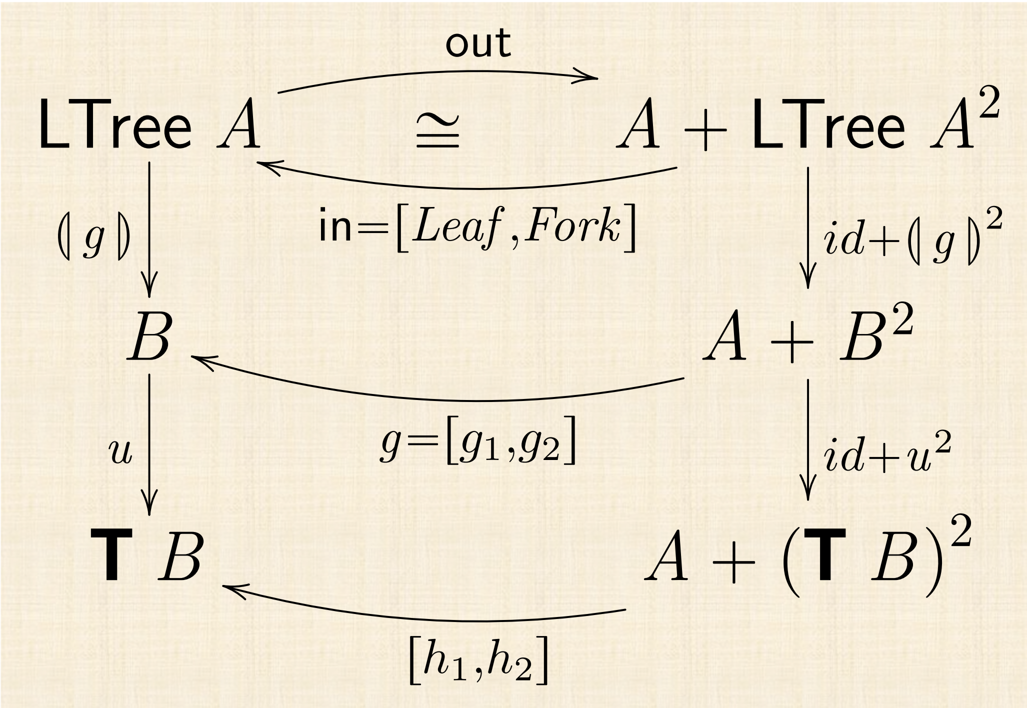
Example – football league





etc

LTree ... catas with monads



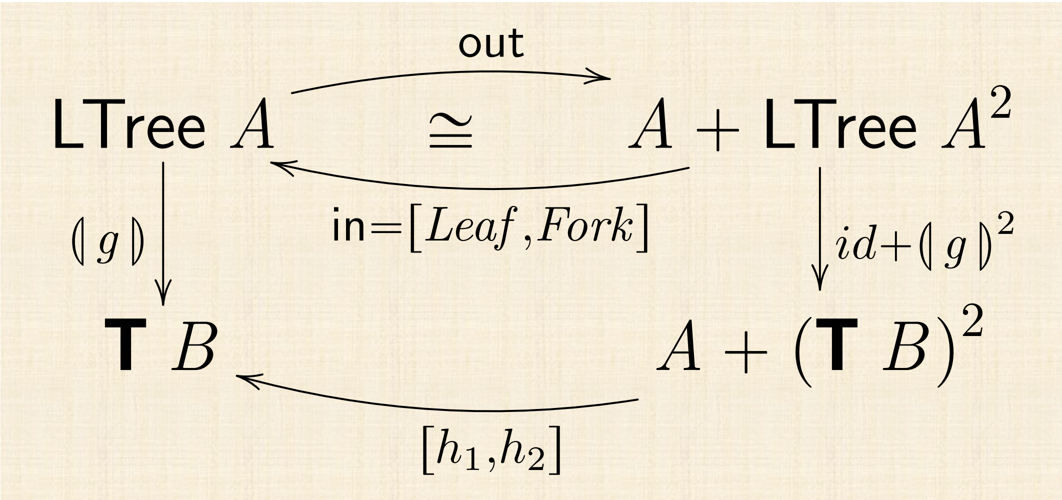
$$\begin{cases} h_1 = u \cdot g_1 \\ h_2 = \mathbf{T}\ g_2 \cdot \delta \end{cases}$$

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

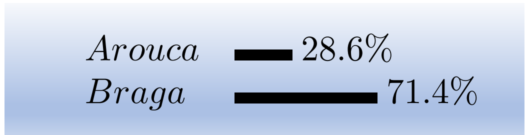
LTree … catas with monads

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

LTree … monadic catas in general



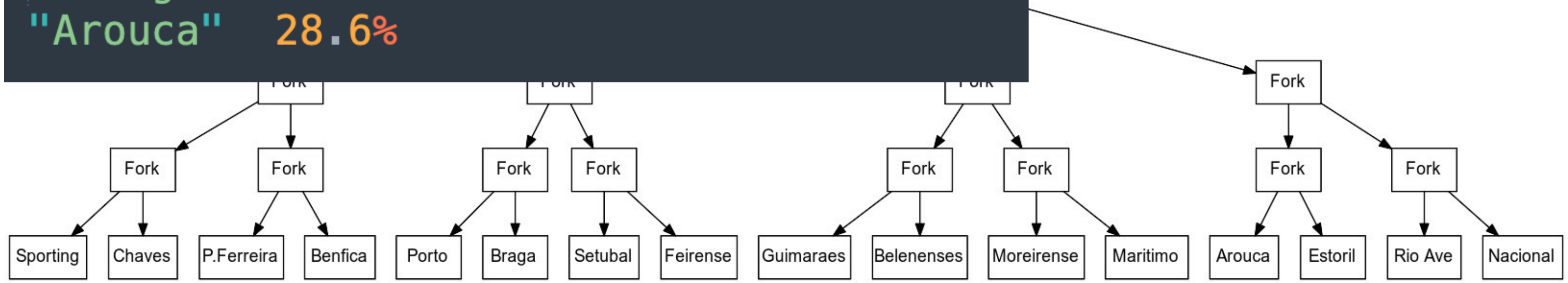
$$f : B^2 \to \mathbf{T}\ B$$

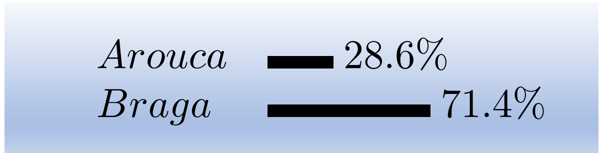


$$h_2\ (x, y) = \mathbf{do}\ \{ a \leftarrow x; b \leftarrow y; f\ (a, b) \}$$

Example – football league

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

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

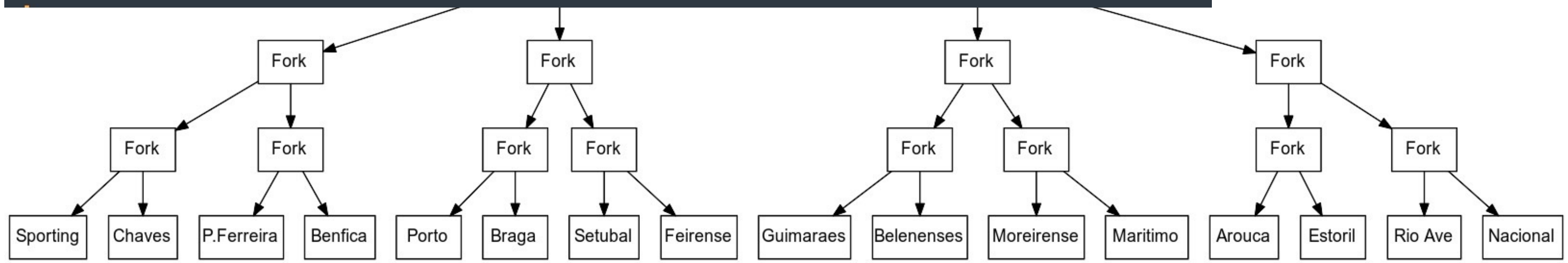




etc

Example – football league

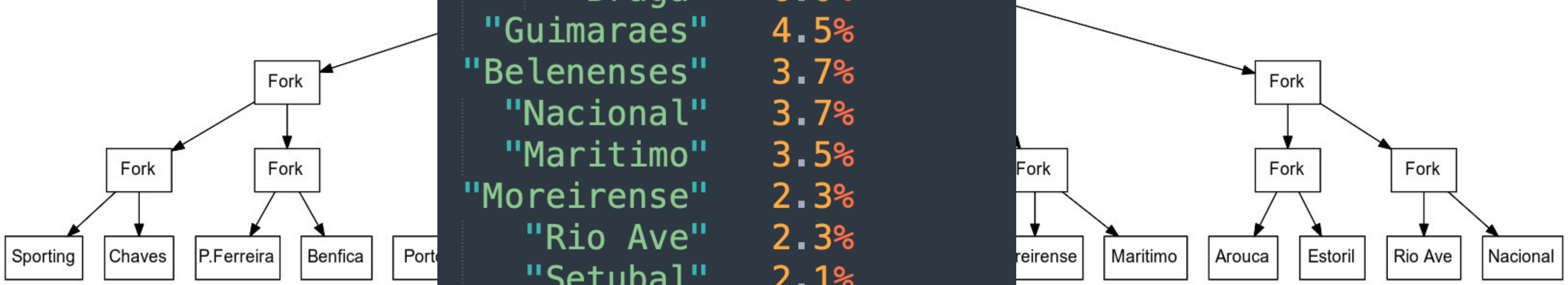
```
h2(d1,d2) = do { a <- d1; b <- d2; jogo(a,b) }

simular  = cataLTree (either return h2)
```




```
calendario = anaLTree lsplit equipas
```

Example – football league

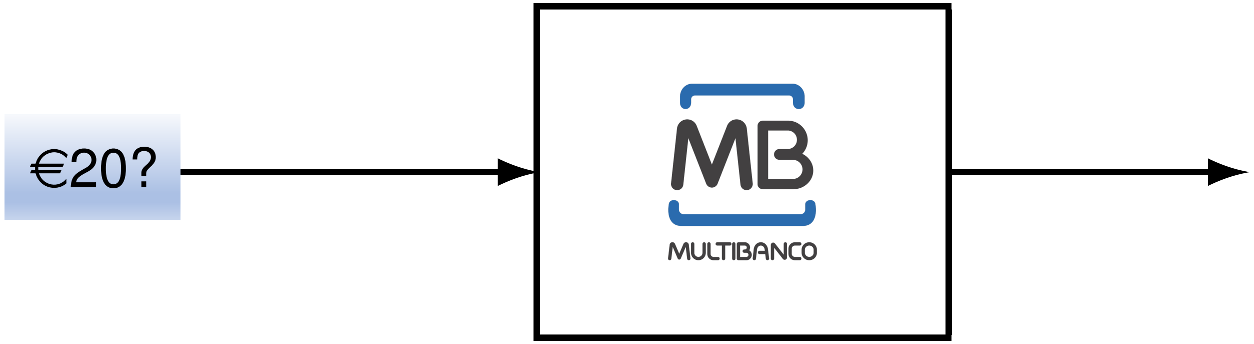
```
*Main> simular calendario
        "Porto"  24.0%
      "Benfica"  22.0%
     "Sporting"  19.8%
        "Braga"   6.0%
    "Guimaraes"   4.5%
   "Belenenses"   3.7%
     "Nacional"   3.7%
     "Maritimo"   3.5%
   "Moreirense"   2.3%
     "Rio Ave"    2.3%
      "Setubal"   2.1%
   "P.Ferreira"  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



etc

STATE MONAD (REACTIVE SYSTEMS)

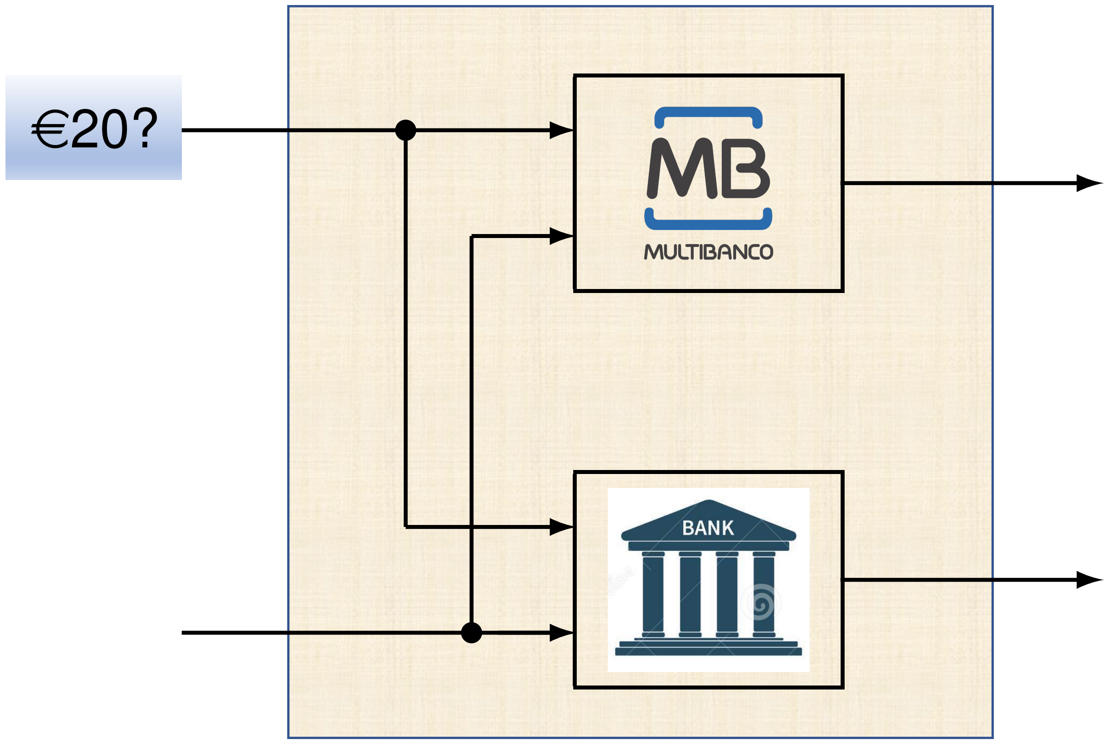
REACTIVE SYSTEMS



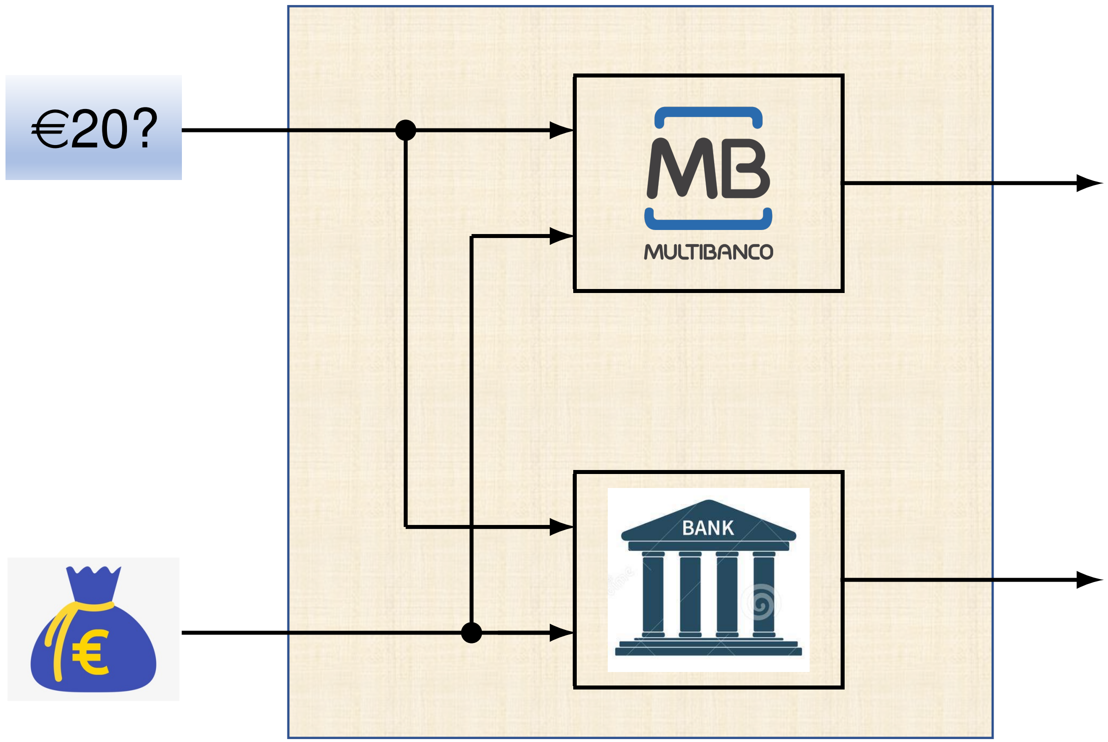
REACTIVE SYSTEMS

REACTIVE SYSTEMS

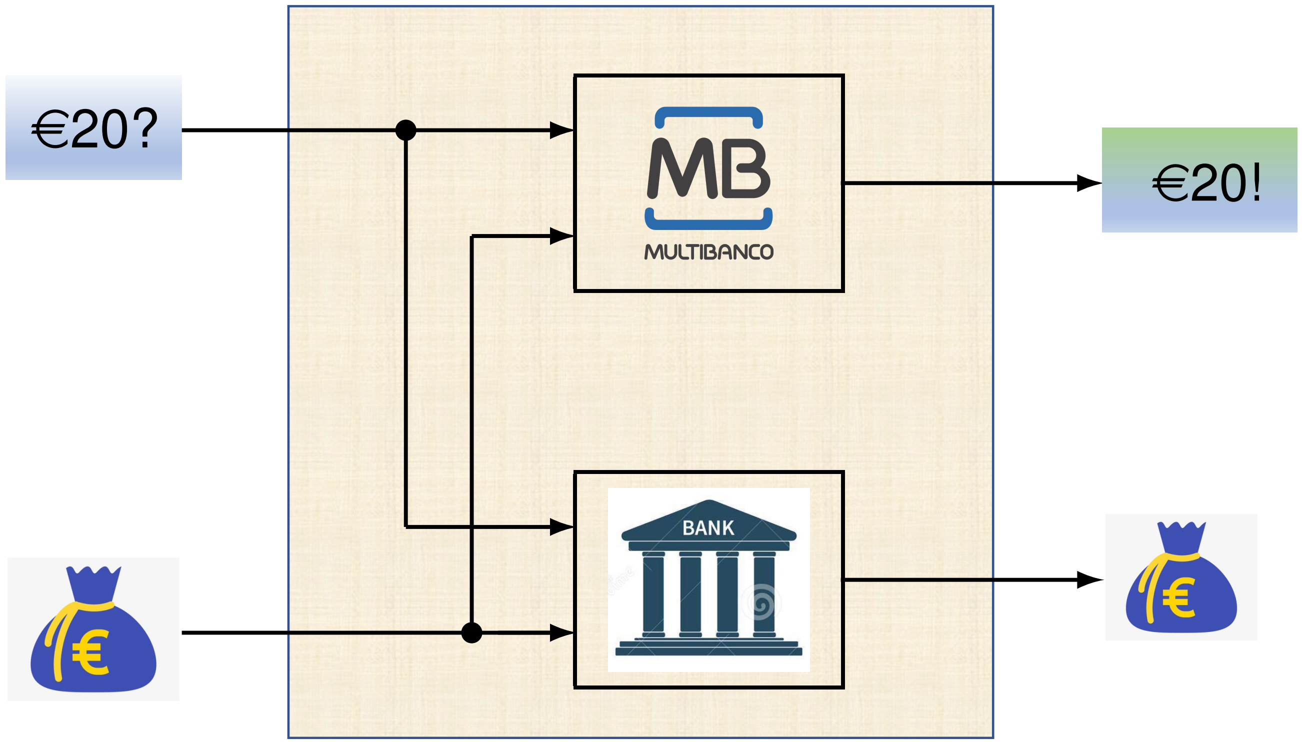
REACTIVE SYSTEMS



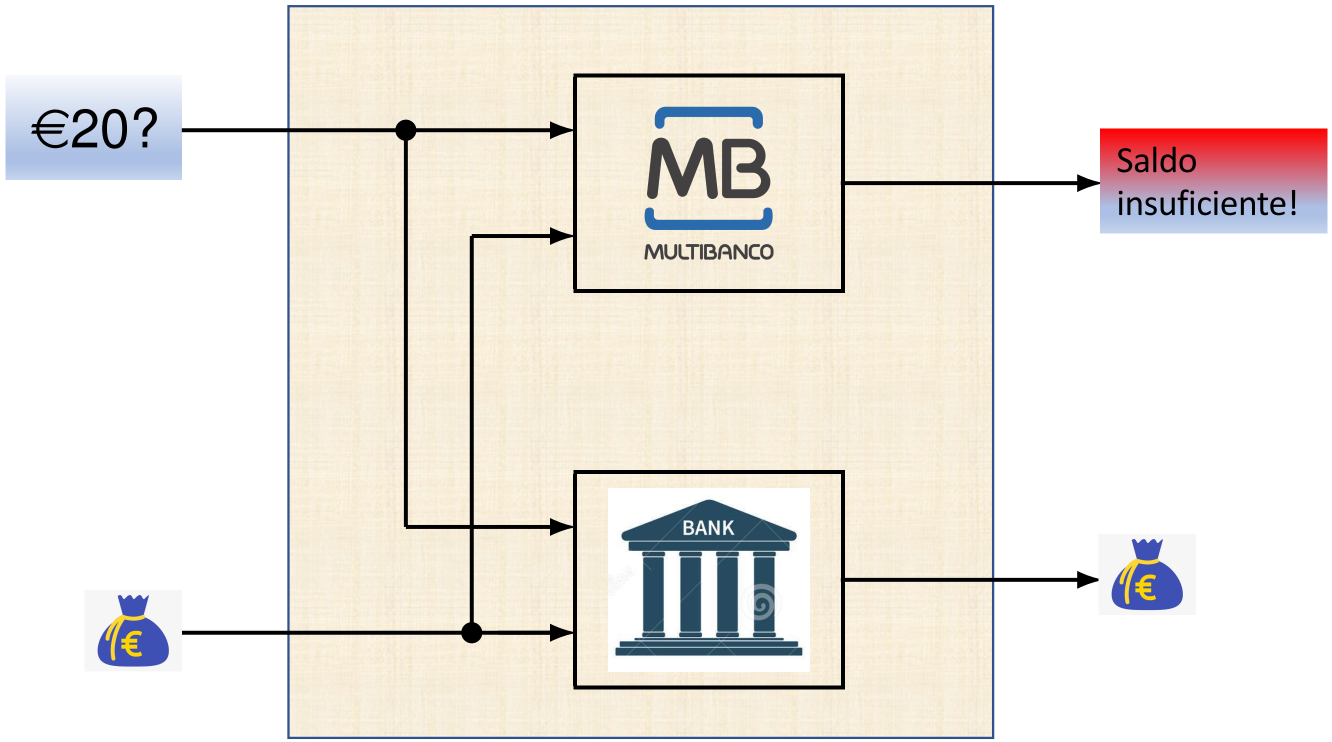
REACTIVE SYSTEMS



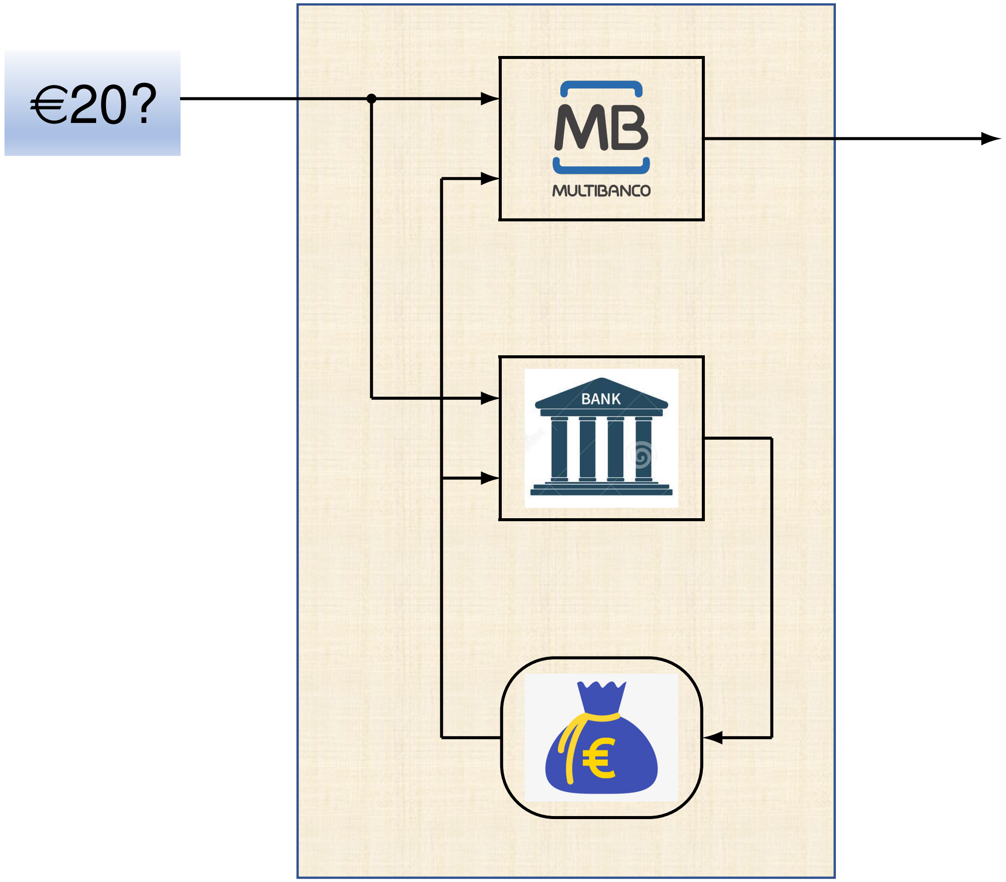
REACTIVE SYSTEMS



REACTIVE SYSTEMS



REACTIVE SYSTEMS

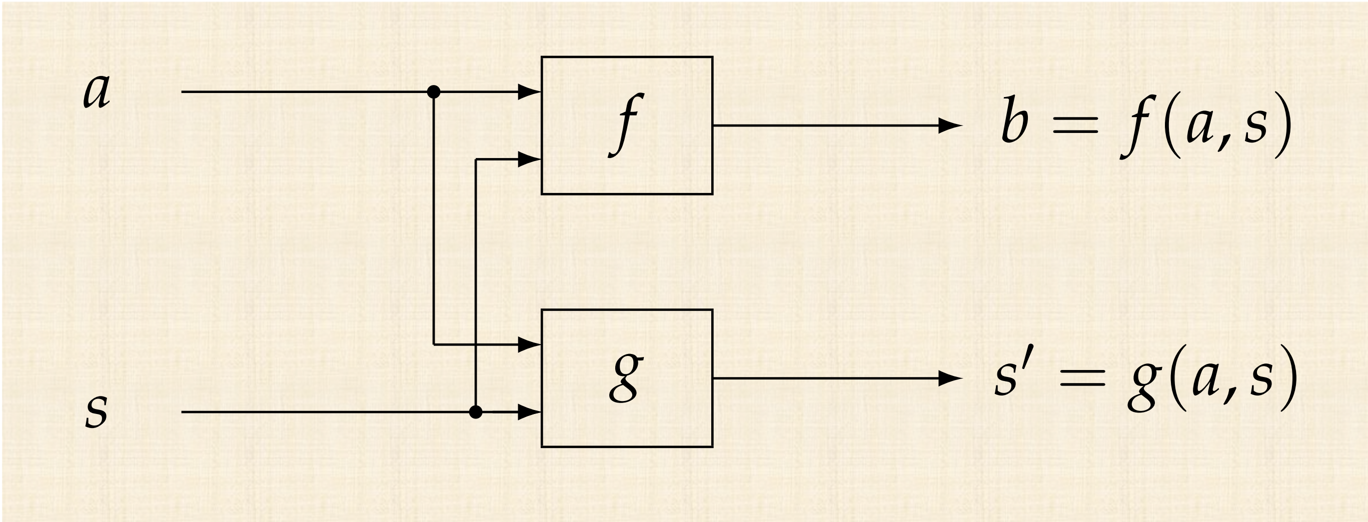


REACTIVE SYSTEMS

$$A \times S \xrightarrow{f} B$$

$$A \times S \xrightarrow{g} S$$

$$A \times S \xrightarrow{\langle f, g \rangle} B \times S$$

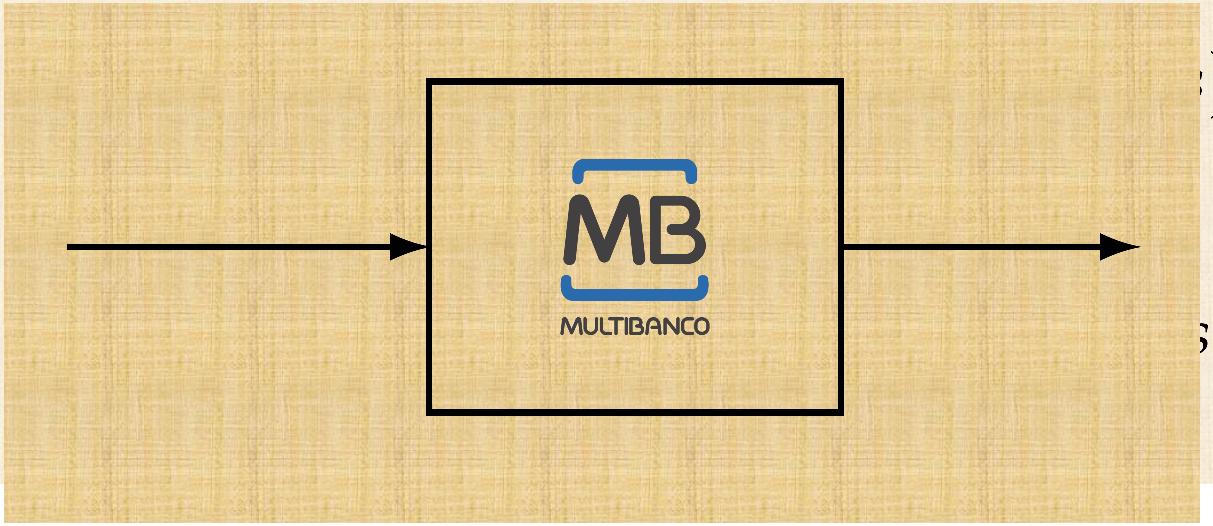


REACTIVE SYSTEMS

$$A \times S \xrightarrow{f} B$$

$$A \times S \xrightarrow{g} S$$

$$A \times S \xrightarrow{\langle f, g \rangle} B \times S$$

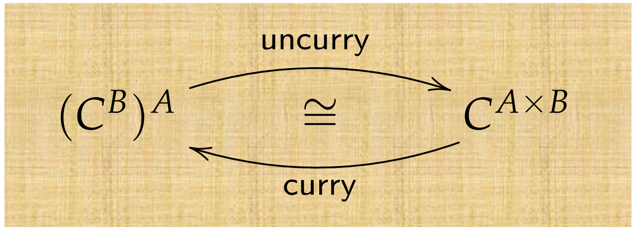


REACTIVE SYSTEMS

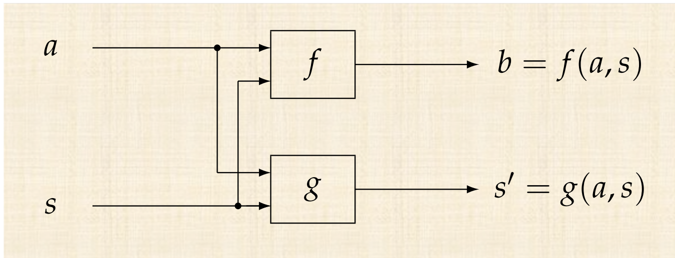
$$\text{curry} :: ((a,b) \to c) \to (a \to b \to c)$$
$$\text{curry}\, f\ a\ b = f\ (a,b)$$

$$A \xrightarrow{f} \ldots$$

$$A \to C^B \quad \cong \quad A \times B \to C$$



$$A \times S \xrightarrow{f} B$$
$$A \times S \xrightarrow{g} S$$
$$A \times S \xrightarrow{\langle f,g \rangle} B \times S$$



REACTIVE SYSTEMS

$$A \times S \xrightarrow{f} B$$

$$A \times S \xrightarrow{g} S$$

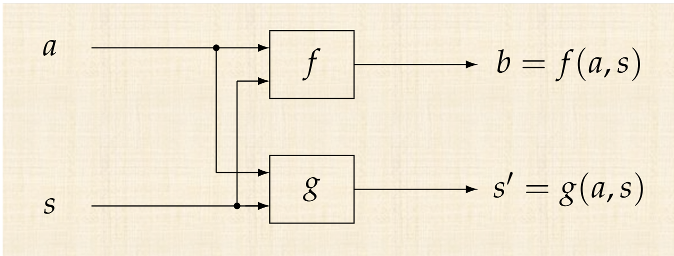
$$A \times S \xrightarrow{\langle f,g \rangle} B \times S$$

$$A \to C^B \quad \cong \quad A \times B \to C$$

$$A \times S \xrightarrow{\langle f,g \rangle} B \times S \quad \cong \quad A \xrightarrow{\overline{\langle f,g \rangle}} (B \times S)^S$$

$$A \xrightarrow{\overline{\langle f,g \rangle}} \underbrace{(B \times S)^S}_{\mathbf{T}\,B}$$

$$A \xrightarrow{\overline{\langle f,g \rangle}} \mathbf{T}\,B$$



STATE MONAD

$$\mathbf{T}\, X = (X \times S)^S$$

$$\mathbf{T}\, f = (f \times id)^S$$

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

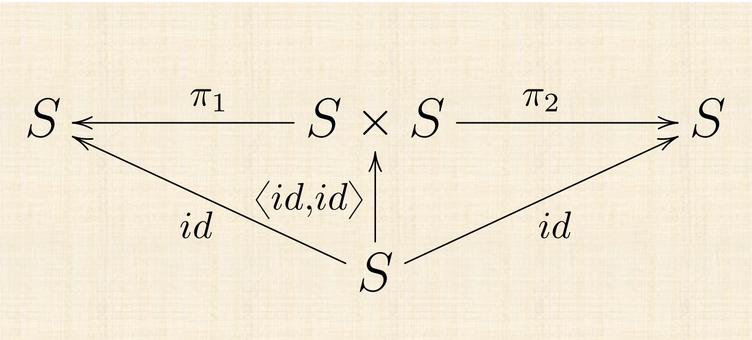
STATE MONAD get

$$\mathbf{T}\,X = (X \times S)^S$$

$$a \in (X \times S)^S$$

$$a = \langle f, g \rangle$$

$$get = \langle id, id \rangle$$



STATE MONAD put

$$\mathbf{T}\,X = (X \times S)^S$$

$$put = \overline{\langle !, \pi_1 \rangle}$$

$$\langle !, \pi_1 \rangle : S \times S \to 1 \times S$$

$$put = \overline{\langle !, \pi_1 \rangle} : S \to (1 \times S)^S$$

STATE MONAD example

$$\mathbf{T}\, X = (X \times S)^S$$

$$
\begin{aligned}
& f\ Empty = return\ Empty \\
& f\ (Node\ (a,(x,y))) = \textbf{do}\ \{ \\
& \quad n \leftarrow get; put\ (n+1); \\
& \quad x' \leftarrow f\ x; \\
& \quad y' \leftarrow f\ y; \\
& \quad return\ (Node\ ((a,n),(x',y')))\,\}
\end{aligned}
$$

$$put = \overline{\langle !, \pi_1 \rangle} : S \to (1 \times S)^S$$

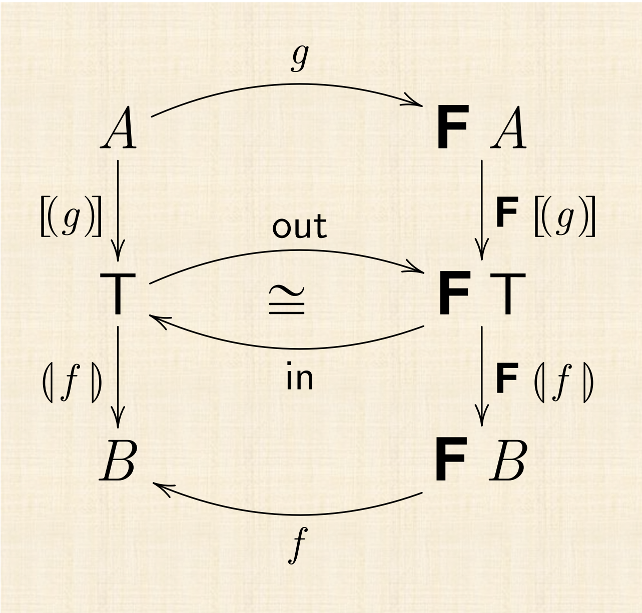
$$get = \langle id, id \rangle$$

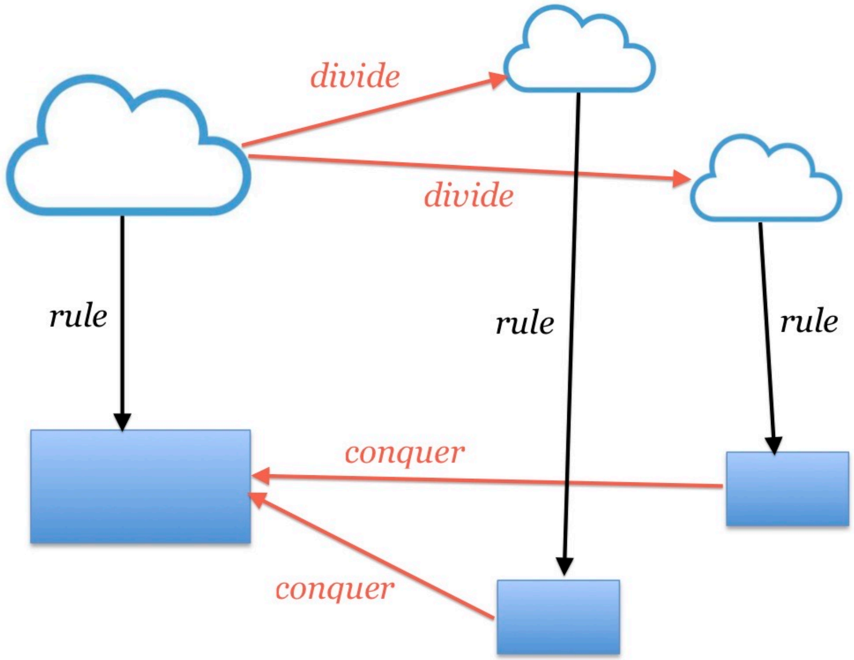
Monad = “racing” functor

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

Epilogue

Back to 'DIVIDE & CONQUER'





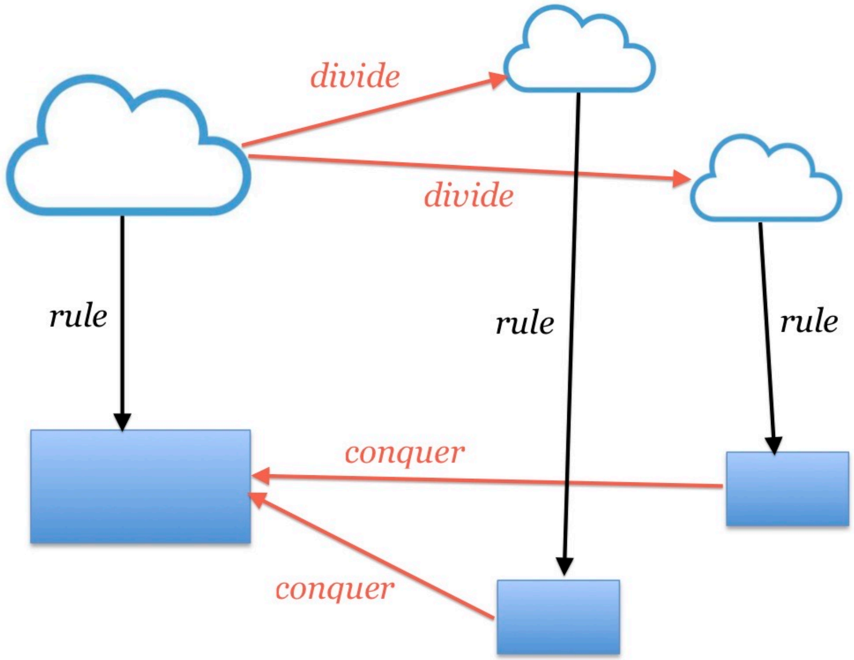
Back to 'DIVIDE & CONQUER'

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

Back to 'DIVIDE & CONQUER'

QUESTION:

In algorithm analysis, how do we find divide and conquer?



Example (FIBONACCI)

$$\mathit{fib}\ 0 = 1$$
$$\mathit{fib}\ 1 = 1$$
$$\mathit{fib}\ (n+2) = \mathit{fib}\ (n+1) + \mathit{fib}\ n$$

$$\mathsf{in} = [\underline{0}, \mathsf{succ}]$$

FIBONACCI

(practical rule)

$$\begin{aligned} &\mathit{fib}\ 0 = 1 \\ &\mathit{fib}\ 1 = 1 \\ &\mathit{fib}\ (n+2) = \mathit{fib}\ (n+1) + \mathit{fib}\ n \end{aligned}$$

$$\begin{aligned} &\mathit{divide}\ 0 = 1 \\ &\mathit{divide}\ 1 = 1 \\ &\mathit{divide}\ (n+2) = \ldots (n+1) \ldots n \end{aligned}$$

$$\begin{aligned} &\mathit{divide}\ 0 = i_1\ 1 \\ &\mathit{divide}\ 1 = i_1\ 1 \\ &\mathit{divide}\ (n+2) = i_2\ (n+1, n) \end{aligned}$$

FIBONACCI

$$\begin{aligned} &\mathit{fib}\ 0 = 1 \\ &\mathit{fib}\ 1 = 1 \\ &\mathit{fib}\ (n+2) = \mathit{fib}\ (n+1) + \mathit{fib}\ n \end{aligned}$$

$$\mathit{divide} : \mathbb{N}_0 \longrightarrow \mathbb{N}_0 + {\mathbb{N}_0}^2$$

$$\begin{aligned} &\mathit{divide}\ 0 = i_1\ 1 \\ &\mathit{divide}\ 1 = i_1\ 1 \\ &\mathit{divide}\ (n+2) = i_2\ (n+1, n) \end{aligned}$$

FIBONACCI

$$fib\ 0 = 1$$
$$fib\ 1 = 1$$
$$fib\ (n+2) = fib\ (n+1) + fib\ n$$

$$divide : \mathbb{N}_0 \to \mathbb{N}_0 + {\mathbb{N}_0}^2$$

$divide$ (

$divide$

$$divide\ (n+2) = \imath_2\ (n+1, n)$$

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

FIBONACCI

$\ldots fib\ (n+1) + fib\ n$

$$\mathsf{T} = \mathsf{LTree}\ \mathbb{N}_0$$

$$divide : \ldots \mathbb{N}_0 + {\mathbb{N}_0}^2$$

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

$divide\ ($ …

$divide$ …

$divide\ (n+2) = \imath_2\ (n+1, n)$

FIBONACCI

$$fib\ 0 = 1$$
$$fib\ 1 = 1$$
$$fib\ (n+2) = fib\ (n+1) + fib\ n$$

$$divide\ 0 = i_1\ 1$$
$$divide\ 1 = i_1\ 1$$
$$divide\ (n+2) = i_2\ (n+1, n)$$

$$\mathsf{T} = \mathsf{LTree}\ \mathbb{N}_0$$

$$conquer : \mathbb{N}_0 + {\mathbb{N}_0}^2 \rightarrow \mathbb{N}_0$$
$$conquer = [id, \mathsf{add}]$$

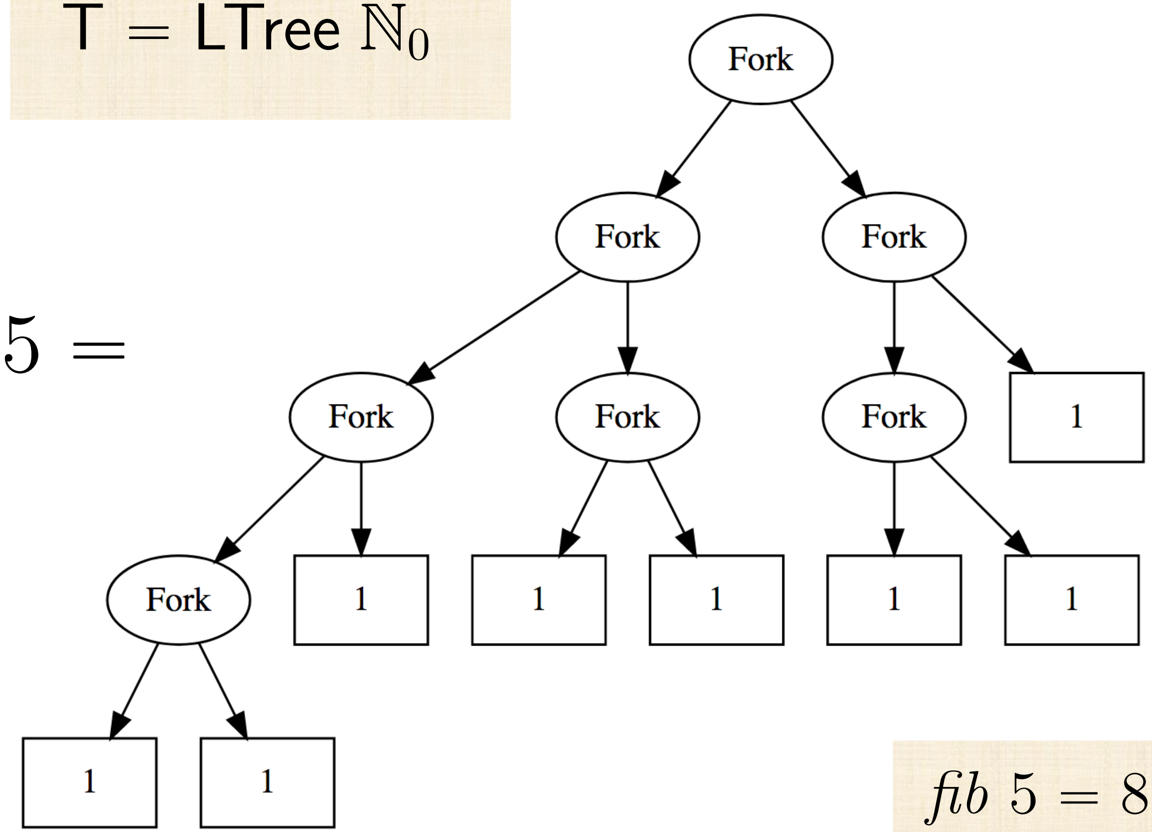
FIBONACCI

```
fib :: (Integral c) => c -> c
fib = hyloL conquer divide
   where
      divide 0 = i1 1
      divide 1 = i1 1
      divide(n+2) = i2(n+1,n)
      conquer = either id add
```

FIBONACCI

$$\mathsf{T} = \mathsf{LTree}\ \mathbb{N}_0$$

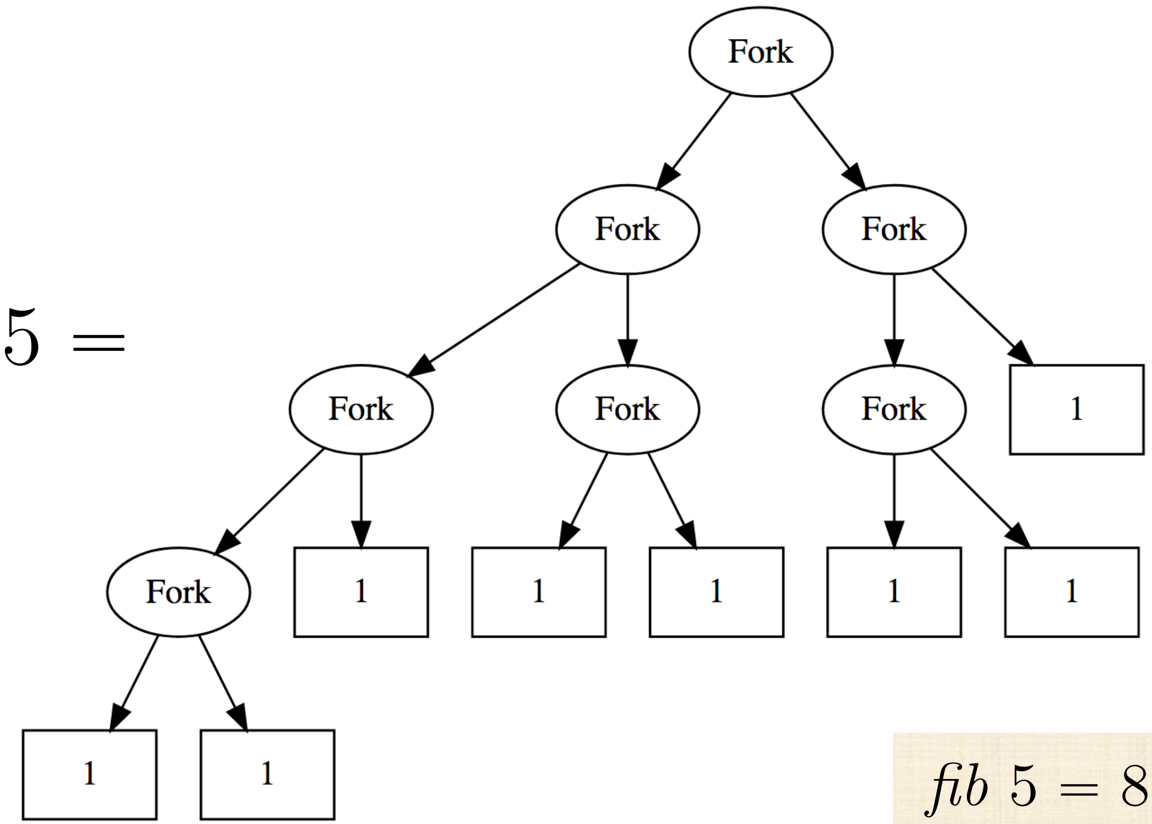
$$[\![(divide)]\!]\ 5 =$$



$$fib\ 5 = 8$$

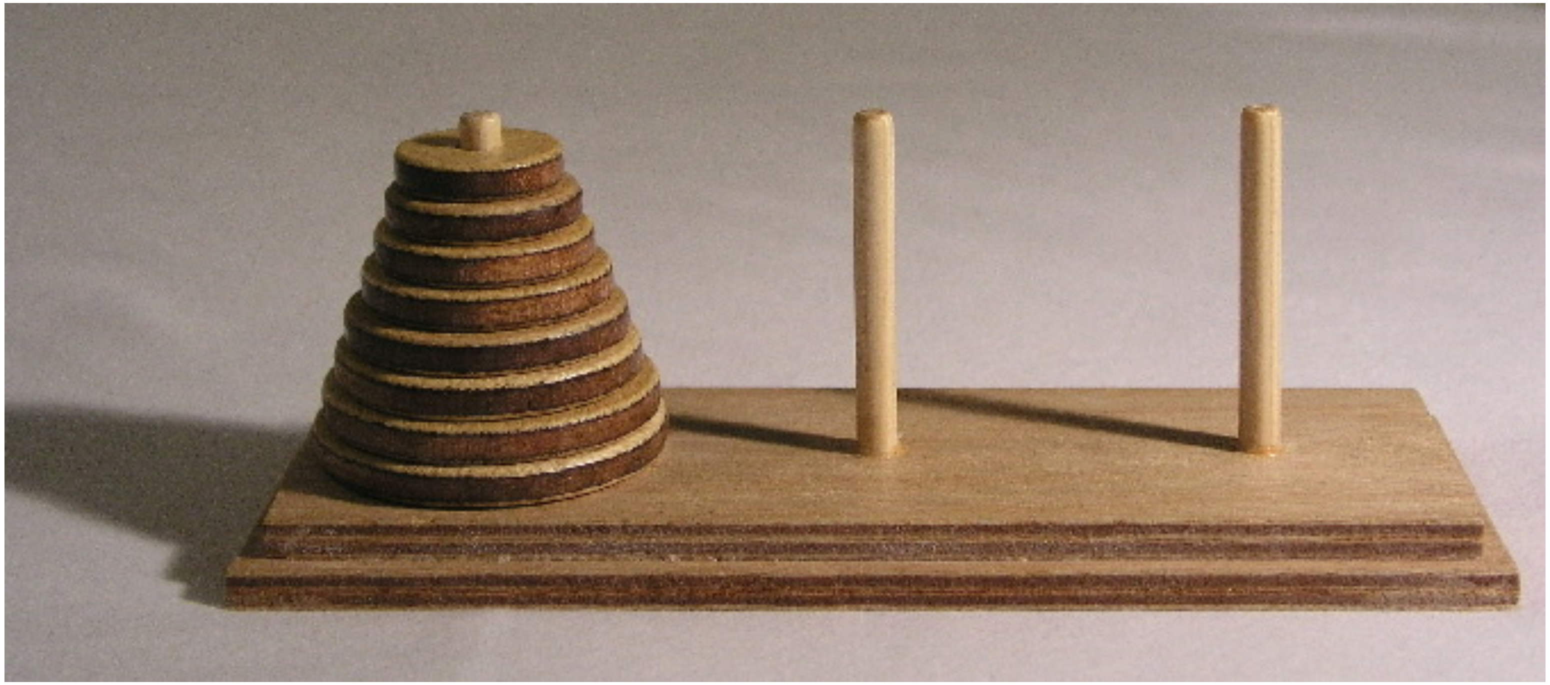
FIBONACCI

$$[\![(divide)]\!]\ 5 =$$

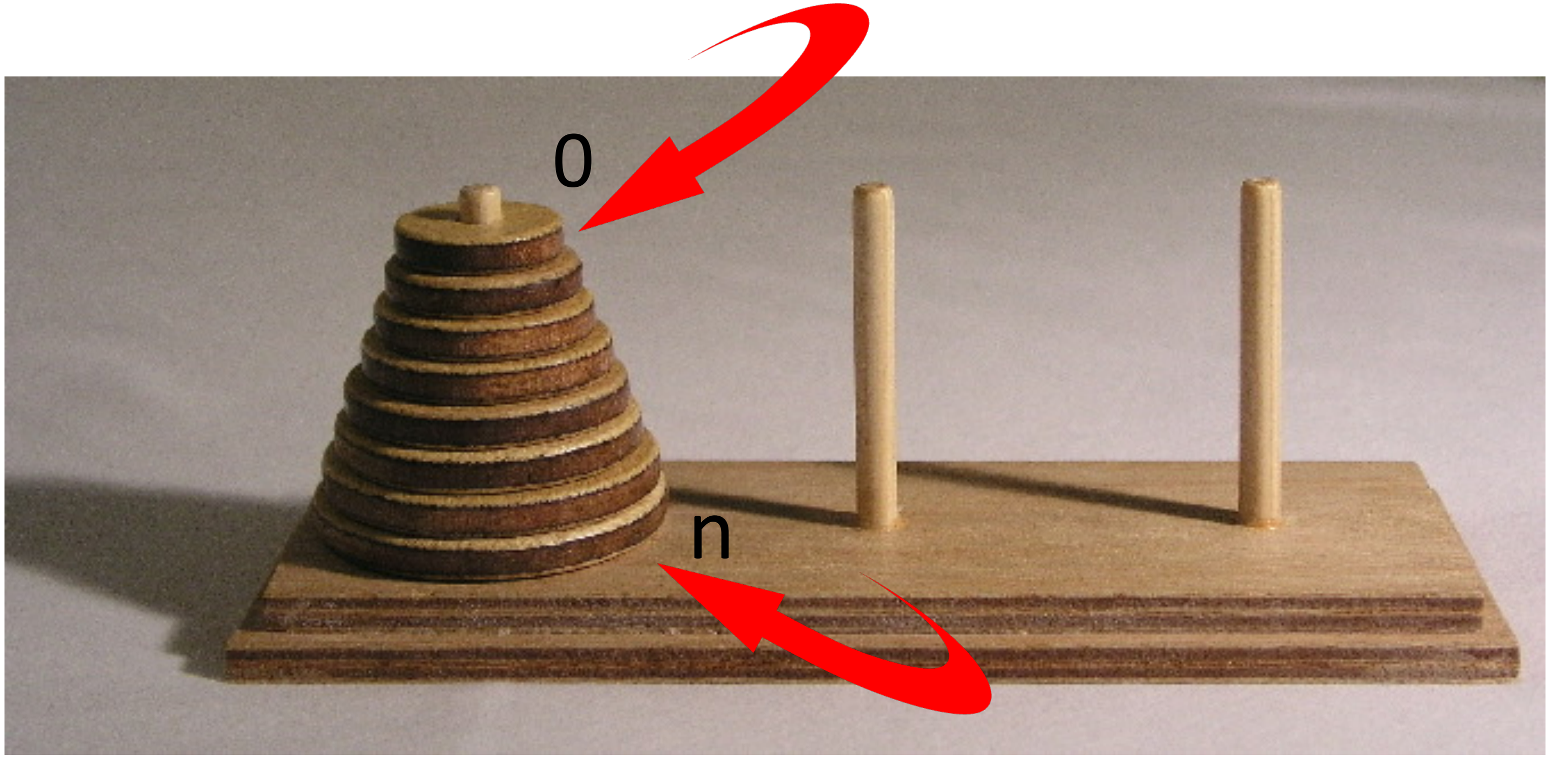


$$\mathit{fib}\ 5 = 8$$

HANOI TOWERS



HANOI TOWERS

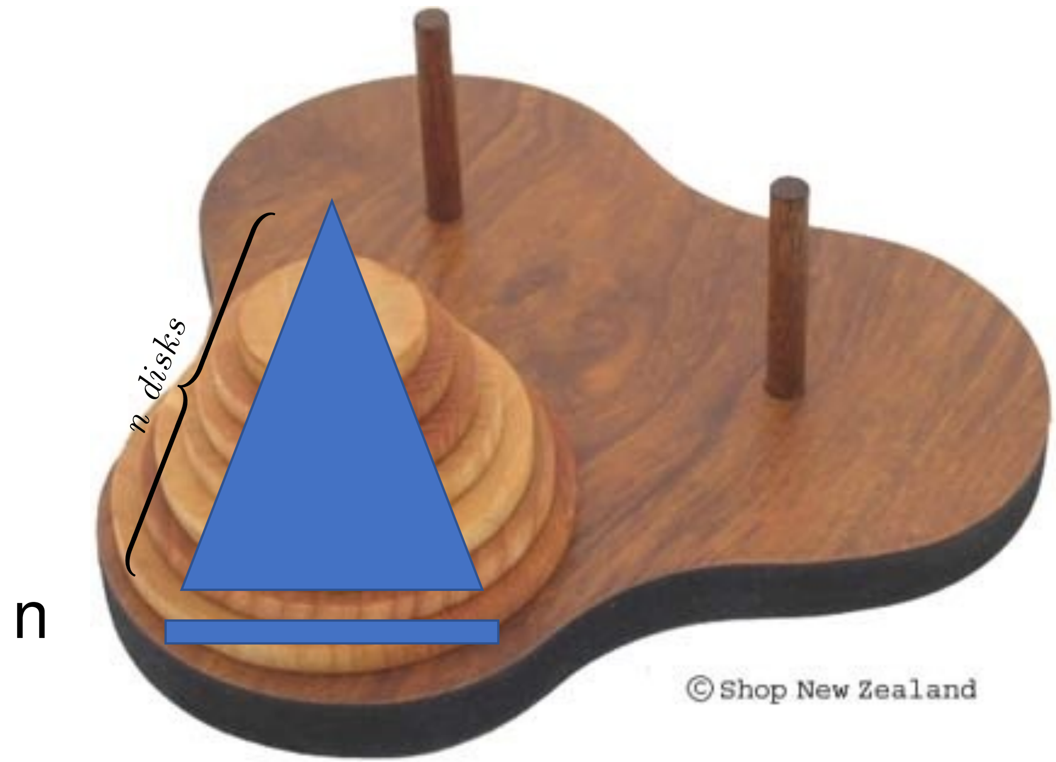


HANOI TOWERS

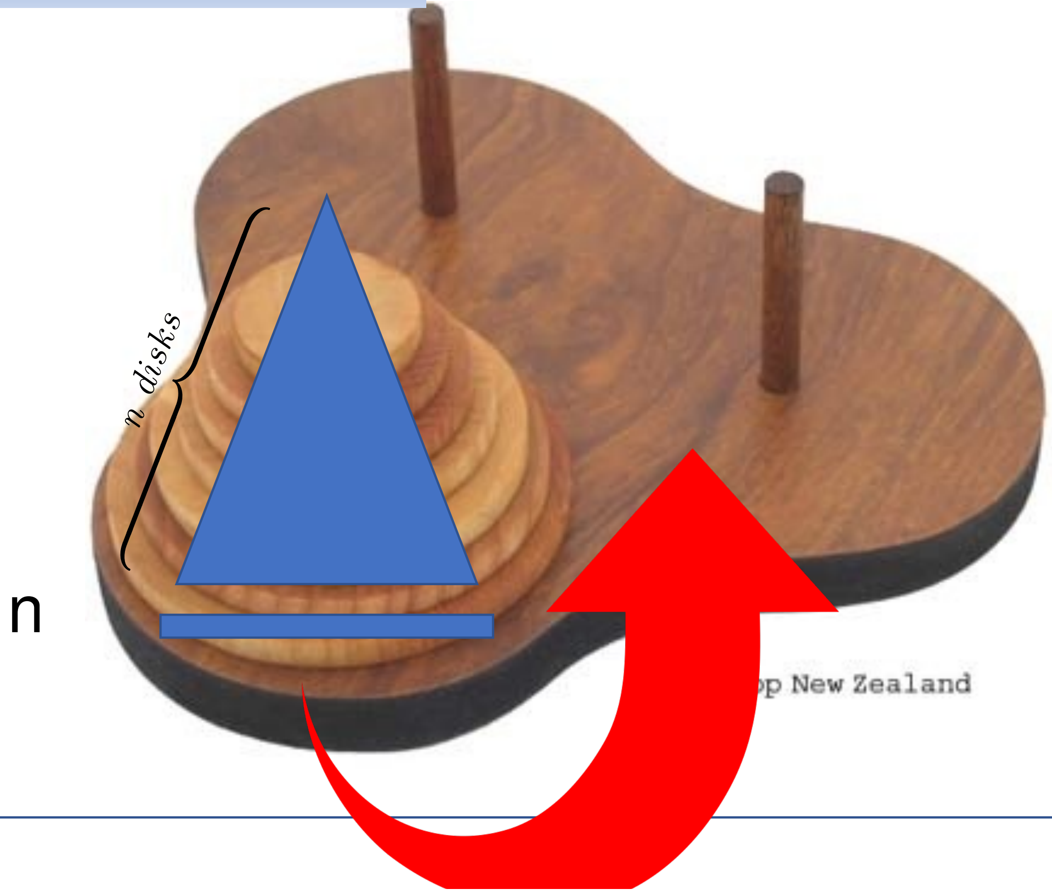
HANOI TOWERS

HANOI TOWERS

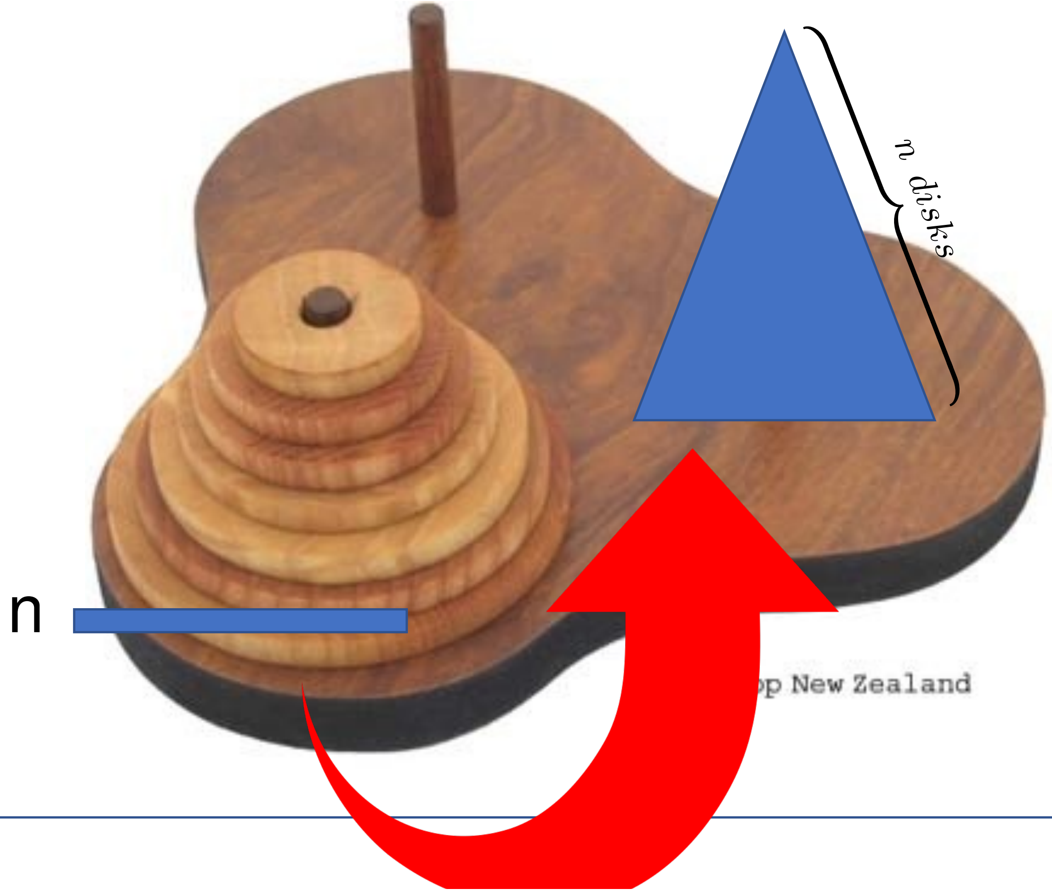
$hanoi\ (d, n+1)$



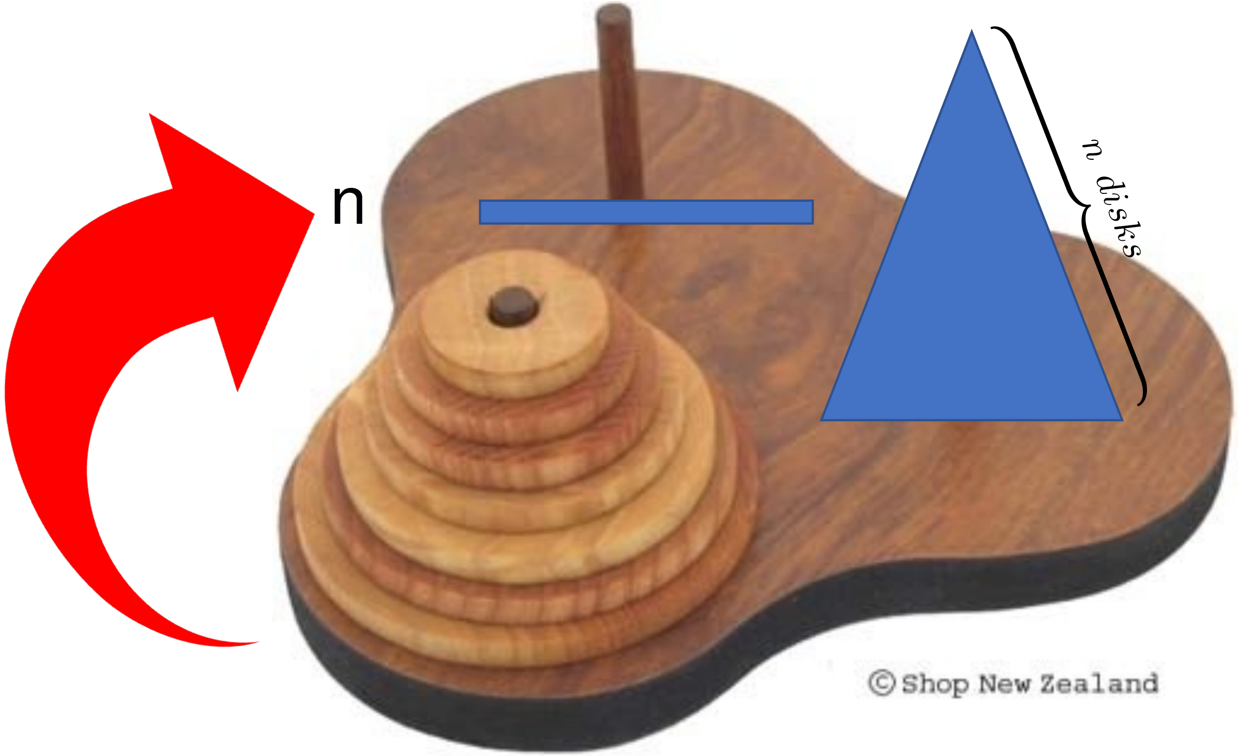
$$hanoi\ (d, n+1) = ...$$

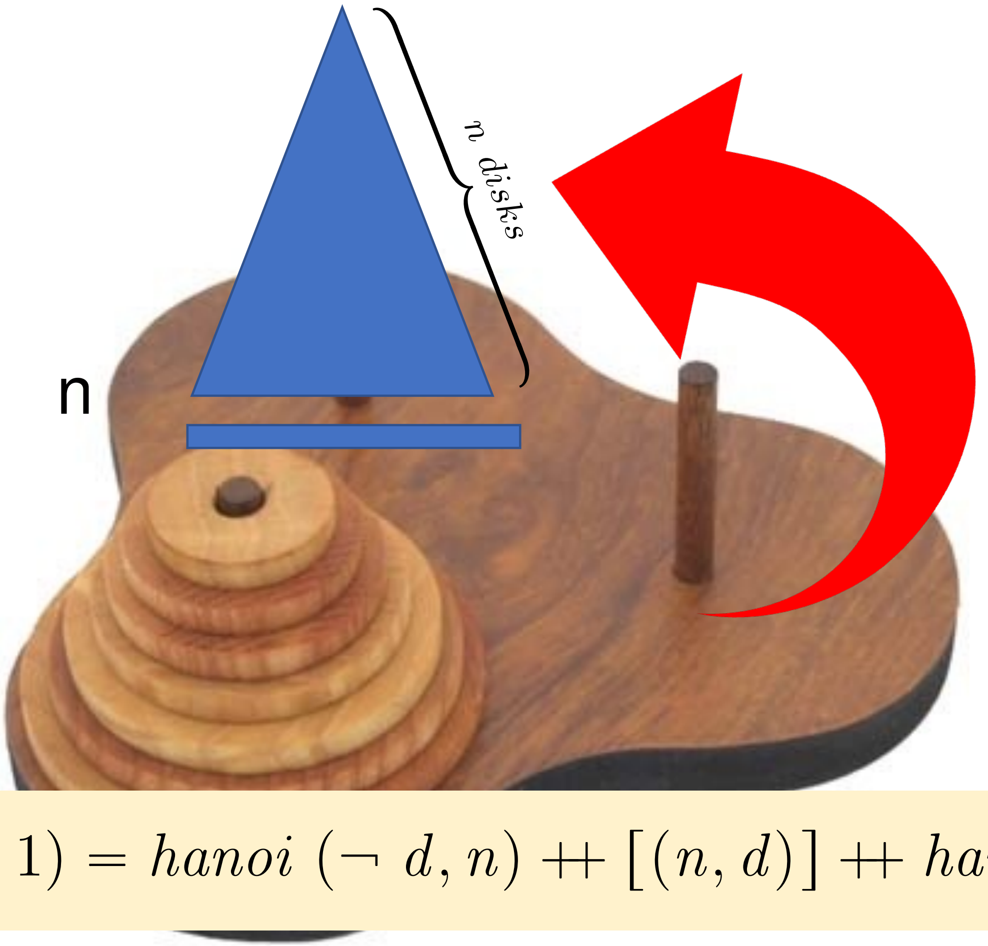


$$hanoi\ (d, n+1) = hanoi\ (\neg\ d, n)$$



$$hanoi\ (d, n+1) = hanoi\ (\neg\ d, n) ++ [(n, d)]$$





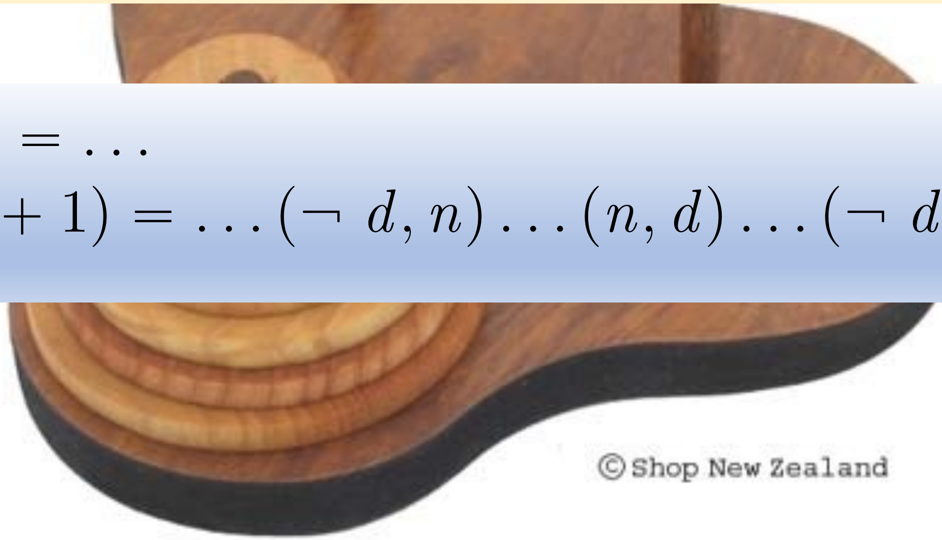
$$hanoi\ (d, n+1) = hanoi\ (\neg\ d, n) +\!\!+ [(n, d)] +\!\!+ hanoi\ (\neg\ d, n)$$

HANOI TOWERS

$$hanoi\ (d, 0) = [\,]$$
$$hanoi\ (d, n+1) = hanoi\ (\neg\ d, n) +\!\!+ [(n, d)] +\!\!+ hanoi\ (\neg\ d, n)$$

HANOI TOWERS

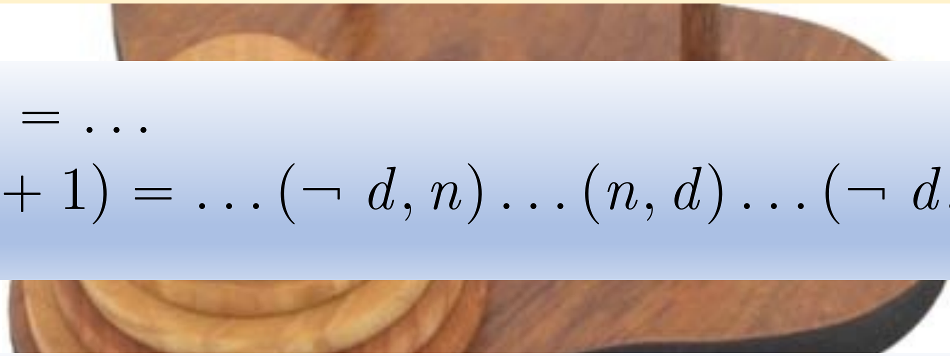
$$hanoi\ (d, 0) = [\,]$$
$$hanoi\ (d, n+1) = hanoi\ (\neg\ d, n) +\!\!+ [(n, d)] +\!\!+ hanoi\ (\neg\ d, n)$$



$$divide\ (d, 0) = \ldots$$
$$divide\ (d, n+1) = \ldots(\neg\ d, n)\ldots(n, d)\ldots(\neg\ d, n)$$

HANOI TOWERS

$$hanoi\ (d, 0) = [\,]$$
$$hanoi\ (d, n+1) = hanoi\ (\neg\ d, n) \mathbin{+\!\!+} [(n, d)] \mathbin{+\!\!+} hanoi\ (\neg\ d, n)$$



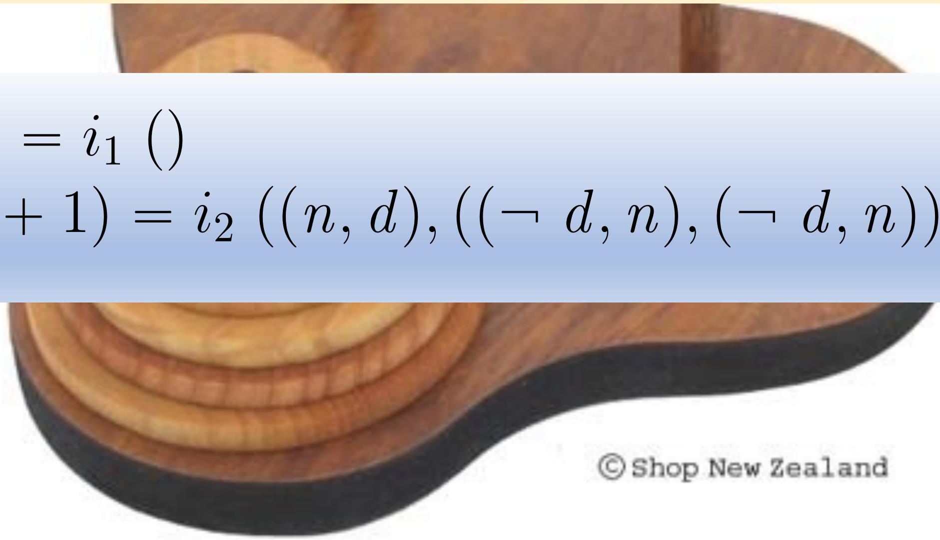
$$divide\ (d, 0) = \ldots$$
$$divide\ (d, n+1) = \ldots(\neg\ d, n)\ldots(n, d)\ldots(\neg\ d, n)$$

$$divide\ (d, 0) = i_1\ ()$$
$$divide\ (d, n+1) = i_2\ (\ldots(\neg\ d, n)\ldots(n, d)\ldots(\neg\ d, n))$$

HANOI TOWERS

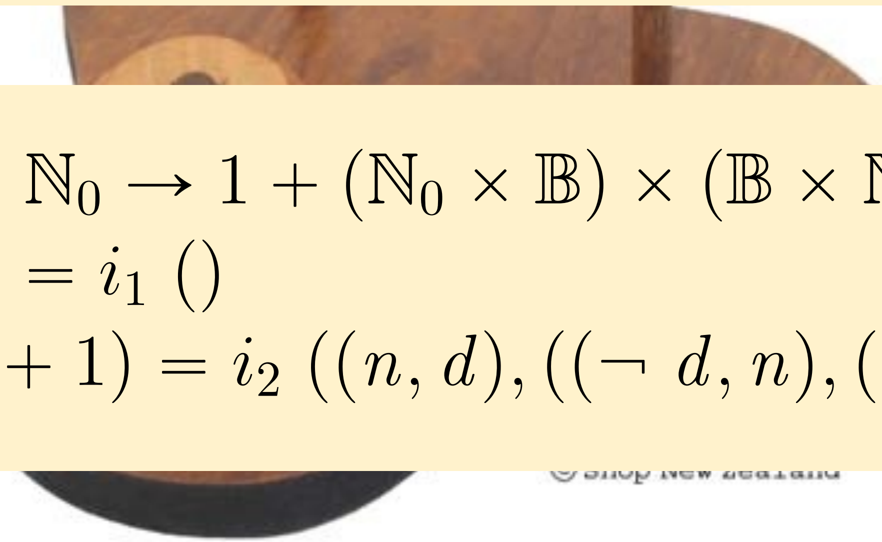
$$
\begin{aligned}
hanoi\ (d, 0) &= [\,] \\
hanoi\ (d, n+1) &= hanoi\ (\neg\ d, n) \mathbin{+\!\!+} [(n, d)] \mathbin{+\!\!+} hanoi\ (\neg\ d, n)
\end{aligned}
$$

$$
\begin{aligned}
divide\ (d, 0) &= i_1\ () \\
divide\ (d, n+1) &= i_2\ ((n, d), ((\neg\ d, n), (\neg\ d, n)))
\end{aligned}
$$



HANOI TOWERS

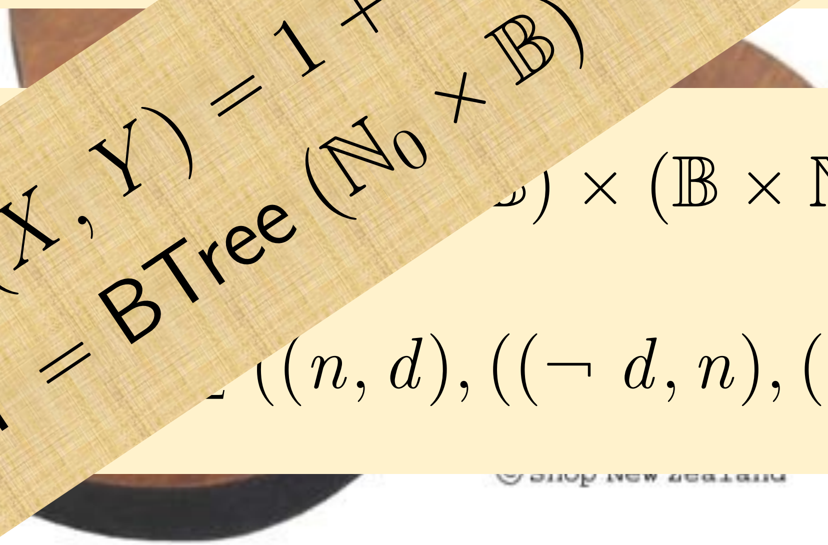
$$hanoi\ (d, 0) = [\,]$$
$$hanoi\ (d, n+1) = hanoi\ (\neg\ d, n) \mathbin{+\!\!+} [(n, d)] \mathbin{+\!\!+} hanoi\ (\neg\ d, n)$$



$$divide :: \mathbb{B} \times \mathbb{N}_0 \to 1 + (\mathbb{N}_0 \times \mathbb{B}) \times (\mathbb{B} \times \mathbb{N}_0)^2$$
$$divide\ (d, 0) = i_1\ ()$$
$$divide\ (d, n+1) = i_2\ ((n, d), ((\neg\ d, n), (\neg\ d, n)))$$

HANOI TOWERS

$$hanoi\ (d, 0) = [\,]$$
$$hanoi\ (d, n+1) = hanoi\ (\neg \ldots noi\ (\neg\ d, n)$$

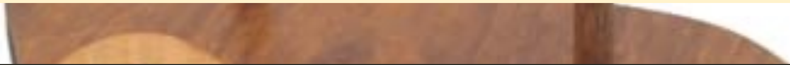


$$divide :: \mathbb{B} \ldots \mathbb{B}) \times (\mathbb{B} \times \mathbb{N}_0)^2$$
$$divide \ldots$$
$$divide\ (\ldots ((n, d), ((\neg\ d, n), (\neg\ d, n)))$$

$$\mathbf{B}\ (X, Y) = 1 + X \times Y^2$$
$$\mathsf{T} = \mathsf{BTree}\ (\mathbb{N}_0 \times \mathbb{B})$$

HANOI TOWERS

$$hanoi\ (d, 0) = [\,]$$
$$hanoi\ (d, n+1) = hanoi\ (\neg\ d, n) + [(n, d)] + hanoi\ (\neg\ d, n)$$



```
hanoi = hyloB conquer divide
   where
     divide(d,0) = i1()
     divide(d,n+1) = i2((n,d),((not d,n),(not d, n)))
     conquer = either nil inord
     inord(a,(x,y)) = x ++ [a] ++ y
```

```haskell
qSort = hyloB conquer divide
    where
      divide []     = i1 ()
      divide (h:t) = i2 (h,(s,l)) where (s,l) = part (<h) t
      conquer = either nil inord
```

```haskell
hanoi = hyloB conquer divide
    where
      divide(d,0) = i1()
      divide(d,n+1) = i2((n,d),((not d,n),(not d, n)))
      conquer = either nil inord
      inord(a,(x,y)) = x ++ [a] ++ y
```

"What for all this"?... **DEMO**

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

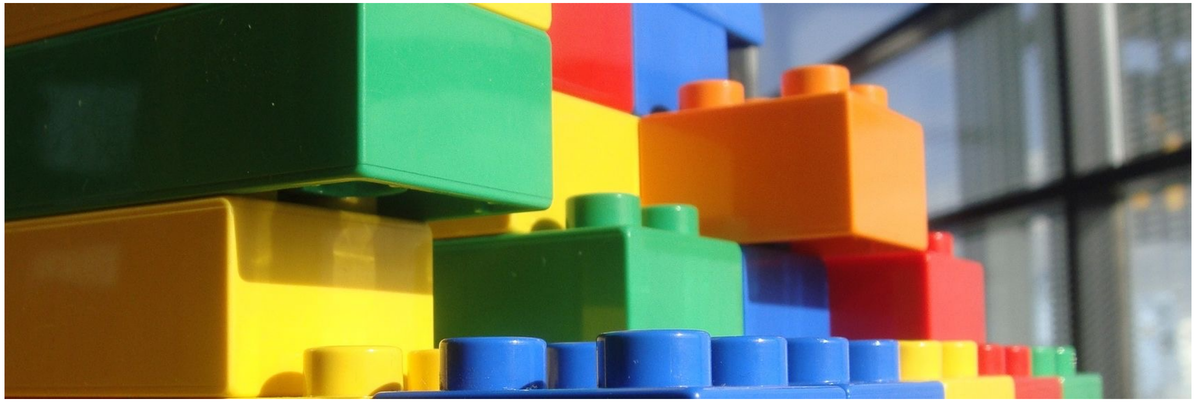
10/03/2014

Romain Jacotin

romain.jacotin@orange.fr

Wrapping up

PROGRAMMING BY CALCULATION



PROGRAMMING BY CALCULATION

PROGRAMMING BY CALCULATION

PROGRAMMING BY CALCULATION

Software Reuse

COPY PASTE

PROGRAMMING

BY

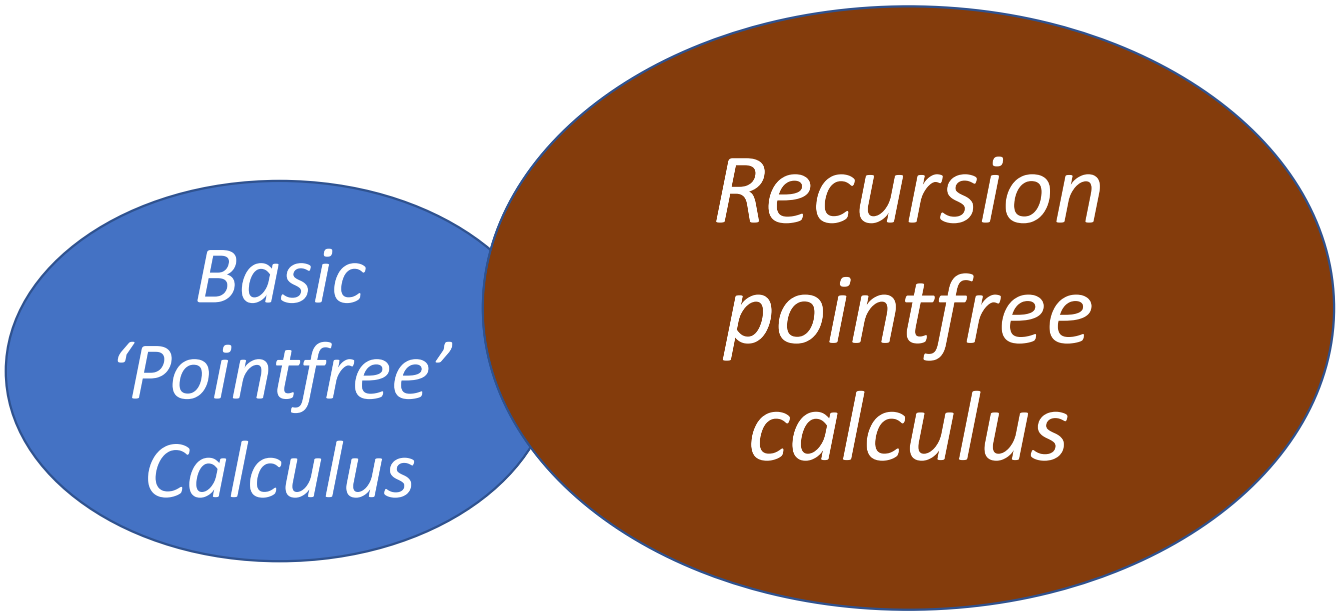
CALCULATION

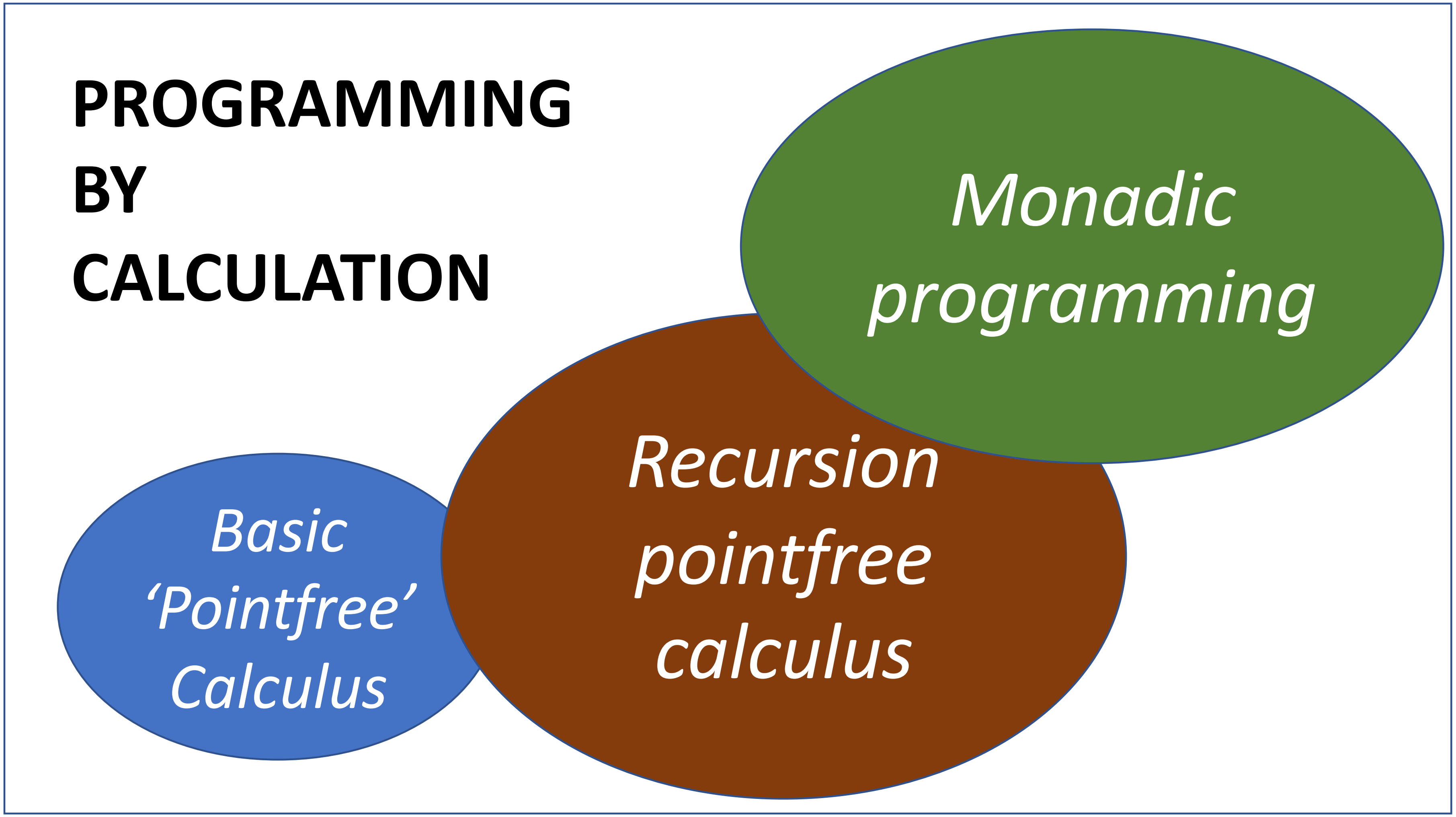
Basic

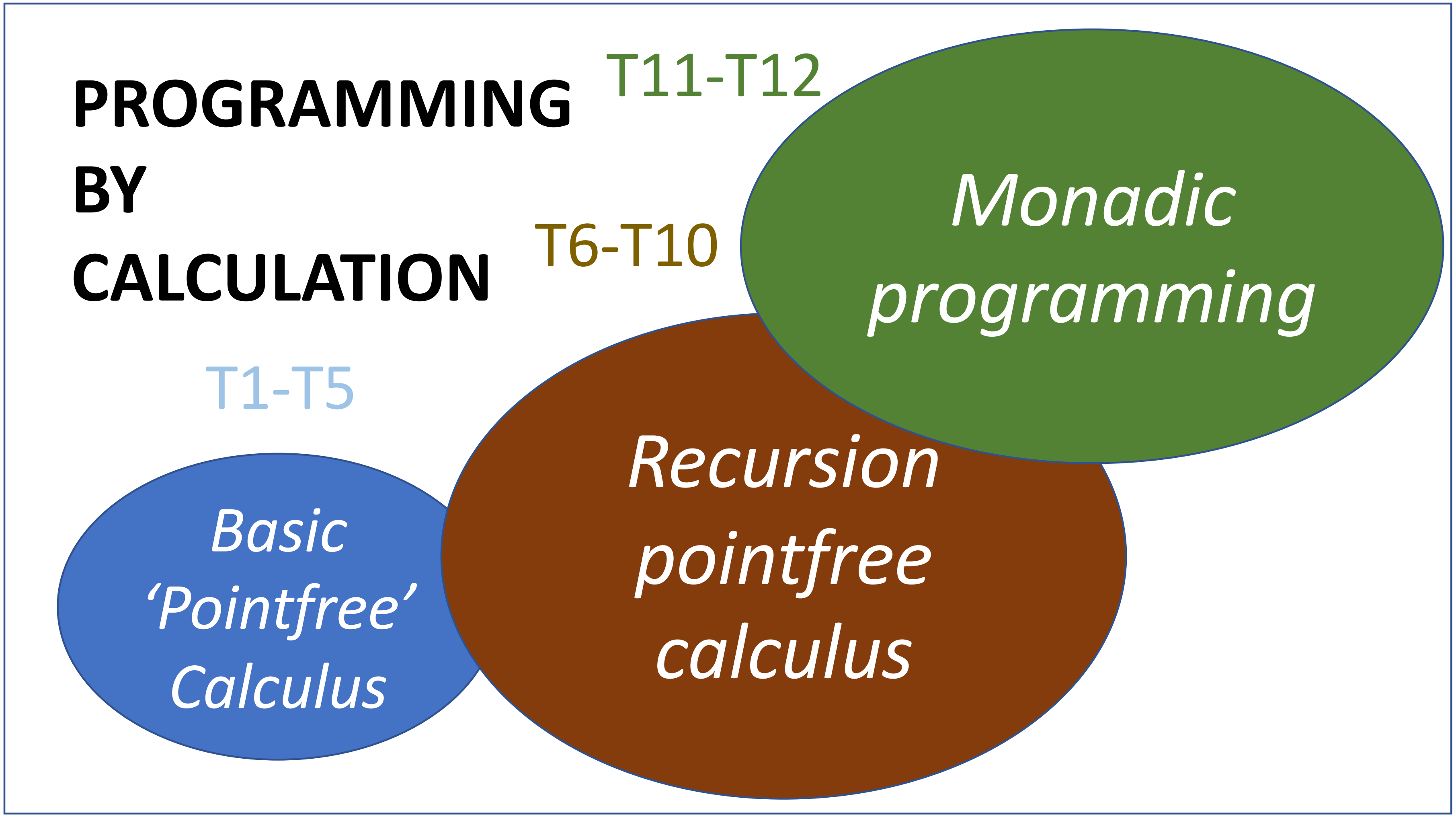
'Pointfree'

Calculus

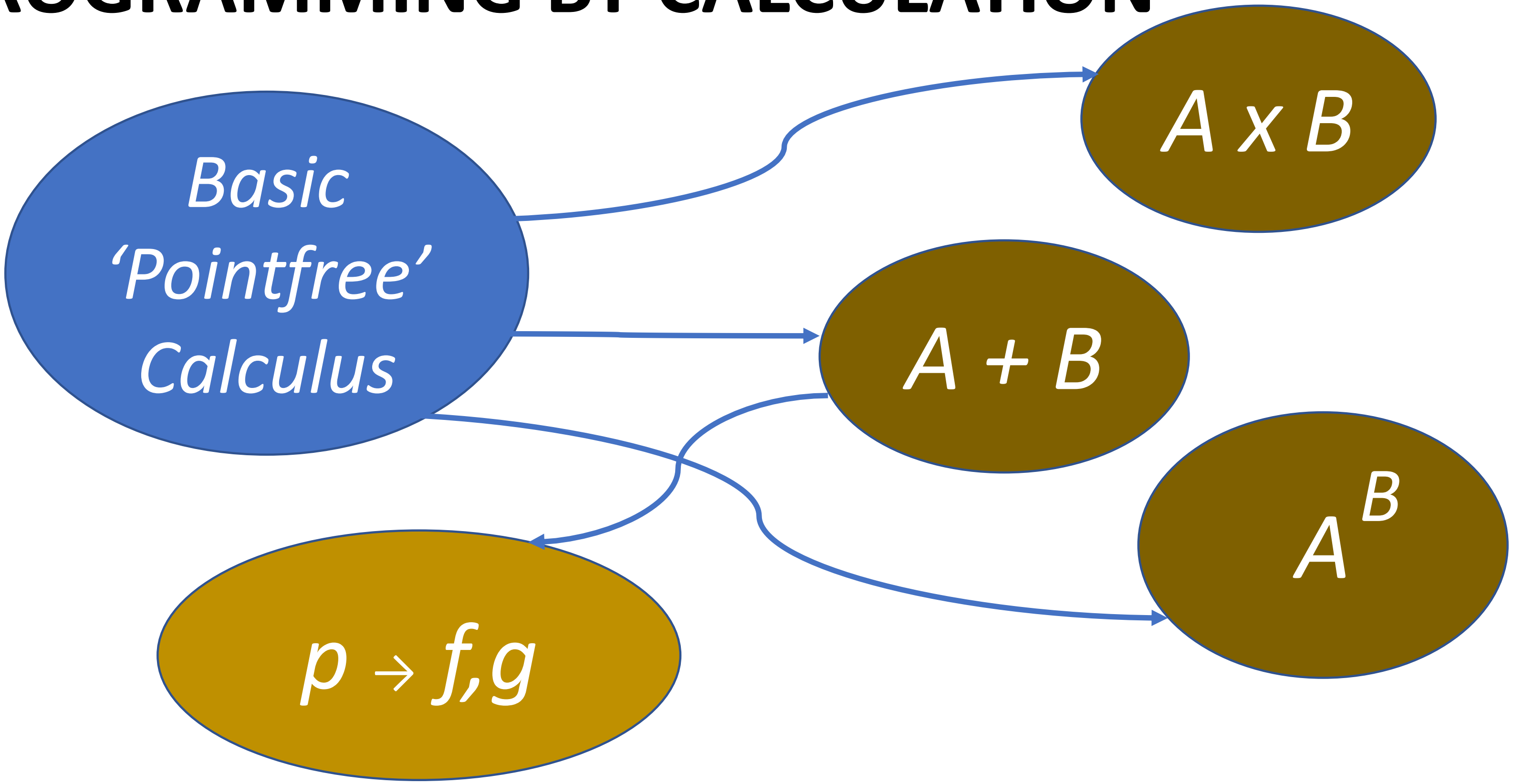
PROGRAMMING BY CALCULATION

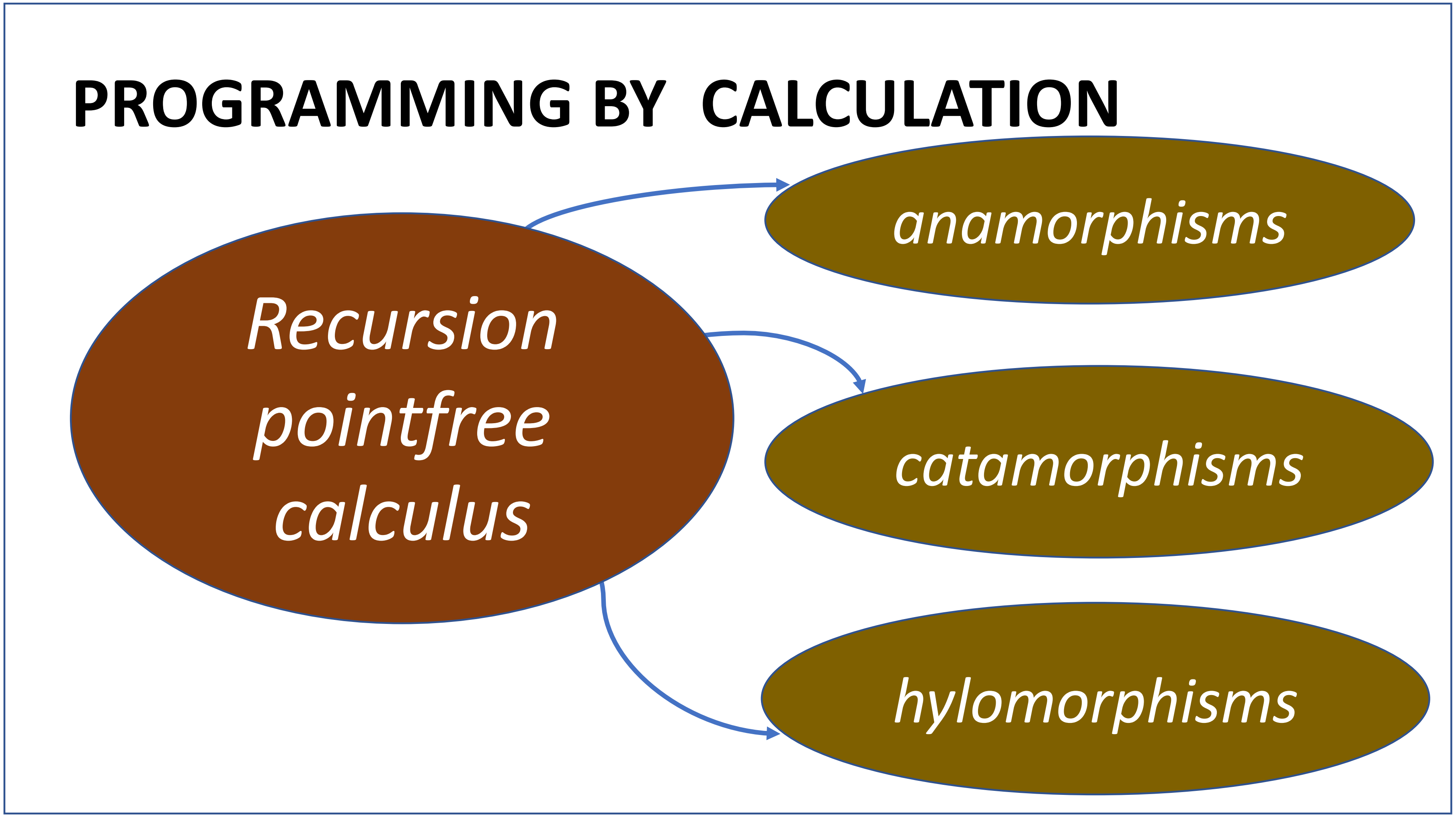



PROGRAMMING
BY
CALCULATION
Monadic programming
Recursion pointfree calculus
Basic 'Pointfree' Calculus

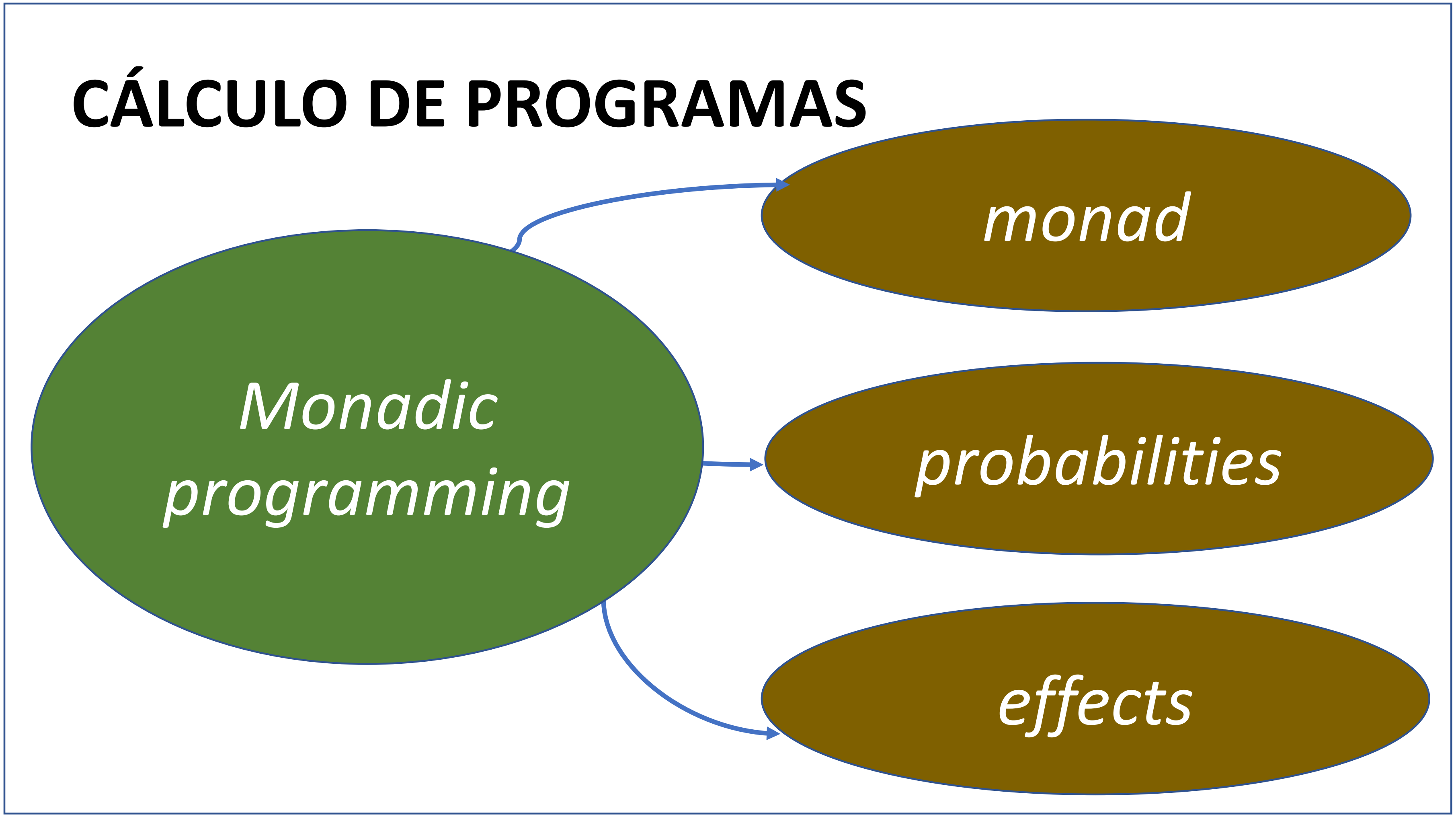

PROGRAMMING
BY
CALCULATION
T11-T12
T6-T10
T1-T5
Monadic programming
Recursion pointfree calculus
Basic ‘Pointfree’ Calculus

PROGRAMMING BY CALCULATION




PROGRAMMING BY CALCULATION
Recursion pointfree calculus
anamorphisms
catamorphisms
hylomorphisms

PROGRAMMING BY CALCULATION


CÁLCULO DE PROGRAMAS
Monadic programming
monad
probabilities
effects

Programming is science, not magic…

Programming is science,

magic.

The END (22/23)