λx.x

Programming with Pure Lambda Calculus

Marvin, GPN22 Karlsruhe

Marvin

Tübingen

Effekt

Bruijn

Common Code

```
function foo(x) {
  if (x > 0)
    return x + foo(x - 1)
  else
    return x
}
```

$\boxed{\texttt{foo(42)}}$

Substitution

```
if (42 > 0)
  return 42 + foo(42 - 1)
else
  return 42
```

Evaluation

return 42 + foo(42 - 1)

Substitution

```
return 42 + {
  if (41 > 0)
    return 41 + foo(41 - 1)
  else
    return 41
}
```

Reduction ≈ Controlled Substitution

(simplified)

Function ≈ Reduction + Evaluation

(simplified)

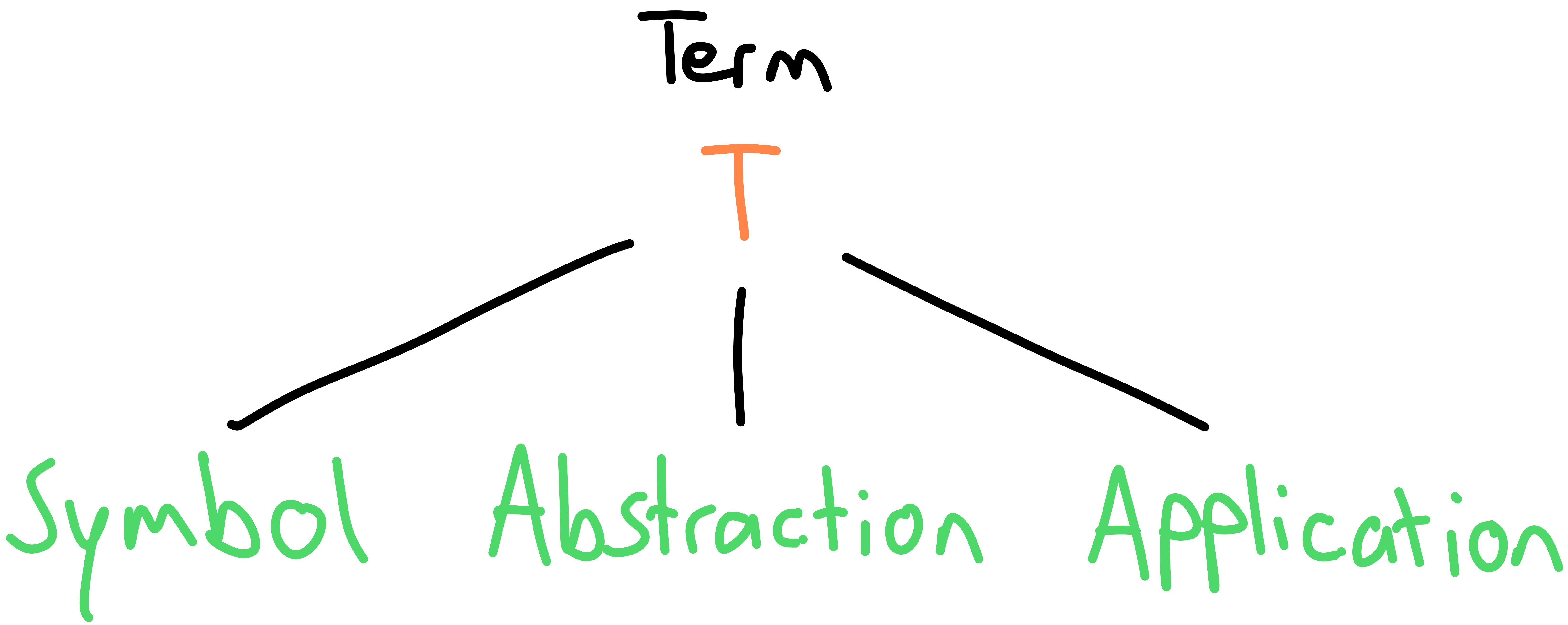
Lambda Function ≈ Reduction + ~~Evaluation~~

(simplified)

Common Code

Different Roots

Lambda Calculus


Term
T
Symbol
Abstraction
Application

Symbol / Variable

$$
\begin{aligned}
\sigma := {} & a\text{-}z \\
\mid {} & \blacksquare, \blacktriangle \\
\mid {} & \ldots
\end{aligned}
$$

Abstraction / Function / Binding

anonym

$\lambda \sigma . T$

Symbol Term

Abstraction / Function / Binding

anonym

$$\lambda \sigma . T \quad \leftrightarrow \quad foo(\sigma) := T$$

Symbol Term

Abstraction / Function / Binding

anonym

$$\lambda \sigma . T \iff \mathrm{foo}(\sigma) := T$$

Symbol Term

$$\mathrm{bar}(x, y) := T \overset{!}{\iff} \lambda x . \lambda y . T$$

Application / Invocation / Call

$(T_1 \ T_2)$

Application / Invocation / Call

$$(T_1 \ T_2) \ \texttt{<->} \ T_1(T_2)$$

Syntax

Term

$$
\begin{aligned}
T \; ::= \;& \lambda \sigma . T \\
\mid \;& (T \; T) \\
\mid \;& \sigma
\end{aligned}
$$

Symbol

$$
\begin{aligned}
\sigma \; ::= \;& a\text{-}z \\
\mid \;& \blacksquare, \blacktriangle \\
\mid \;& \ldots
\end{aligned}
$$

Term

$$T ::= \lambda\sigma.T \mid (T\ T) \mid \sigma$$

Symbol

$$\sigma ::= a\text{-}z \mid \blacksquare, \blacktriangle \mid \ldots$$

$$(\ldots((T_1\ T_2)\ T_3)\ldots) = (T_1\ T_2\ T_3 \ldots)$$

e.g.

$$(((a\ b)\ c)\ d) = (a\ b\ c\ d)$$

$$((a\ (b\ c))\ d) = (a\ (b\ c)\ d)$$

Term

$$T := \lambda \sigma . T$$
$$\mid (T\ T)$$
$$\mid \sigma$$

Symbol

$$\sigma := a\text{-}z$$
$$\mid \blacksquare, \blacktriangle$$
$$\mid \dots$$

Valid

- $\lambda x . x$
- $\lambda x . (x\ x\ x)$
- $(a\ \lambda b . b)$
- a
- $\lambda a . \lambda b . (a\ b)$

Term

$$T ::= \lambda\sigma.T \;\mid\; (T\ T) \;\mid\; \sigma$$

Symbol

$$\sigma ::= a\text{-}z \;\mid\; ■, ▲ \;\mid\; \ldots$$

Valid

- $\lambda x.x$
- $\lambda x.(x\ x\ x)$
- $(a\ \lambda b.b)$
- a

- $\lambda a.\lambda b.(a\ b)$
- [$\lambda ab.(a\ b)$]

Term

$$
\begin{aligned}
T := \; & \lambda \sigma . T \\
\mid \; & (T \; T) \\
\mid \; & \sigma
\end{aligned}
$$

Symbol

$$
\begin{aligned}
\sigma := \; & a\text{-}z \\
\mid \; & \blacksquare, \blacktriangle \\
\mid \; & \ldots
\end{aligned}
$$

Invalid

- $\lambda(a \; b)$
- $x. x$
- 42
- 'a'
- "gpn"

Term

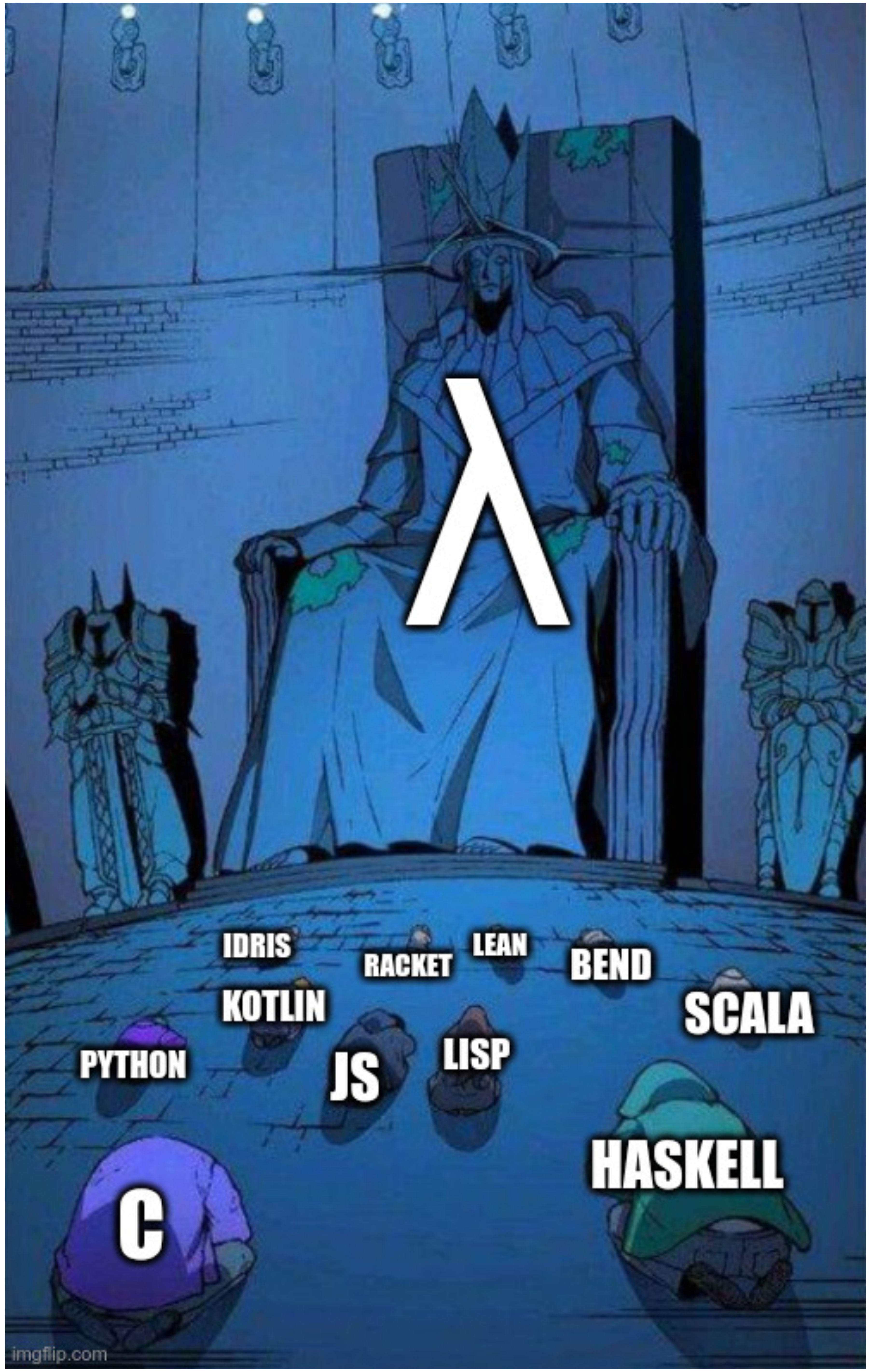
$$T := \lambda\sigma.T \;|\; (T\;T) \;|\; \sigma$$

Symbol

$$\sigma := a\text{-}z \;|\; \blacksquare, \blacktriangle \;|\; \ldots$$

$\lambda x . \lambda y . (x\ y)$

- JavaScript: „x => y => x(y)“
- Python: „lambda x.lambda y.x(y)“
- Haskell: „\x -> \y -> x y“



Substitution / Reduction

- $(\lambda x.x \ T) \rightsquigarrow T$

vs. $f(x) := x$

$\rightarrow f(T) = T$

Substitution / Reduction

- $(\lambda x. x \; T) \rightsquigarrow T$
- $(\lambda x. \lambda y. x \; T) \rightsquigarrow \lambda y. T$

vs.

$f(x) := x$

$\rightarrow f(T) = T$

Substitution / Reduction

- $(\lambda x.x\ T) \leadsto T$ vs. $f(x) := x$ $\to f(T) = T$
- $(\lambda x.\lambda y.x\ T) \leadsto \lambda y.T$
- $(\lambda x.(x\ x)\ T) \leadsto (T\ T)$

$\hookrightarrow (\lambda x.(x\ x)\ \lambda y.(y\ y))$

Evaluation == Redex Hunt!

$$(\underbrace{\lambda\sigma.\,T_1\ T_2}_{\text{Redex}}) \rightsquigarrow T_1[\sigma = T_2]$$

Evaluation == Redex Hunt!

$$(\underbrace{\lambda \sigma. T_1\ T_2}_{\text{Redex}}) \rightsquigarrow T_1[\sigma = T_2]$$

Controlled Redex Hunt

$$[\lambda \sigma . T] \rightsquigarrow \lambda \sigma . [T]$$

$$! \quad [(\lambda \sigma . T_1 \; T_2)] \rightsquigarrow [T_1[\sigma = T_2]] \quad !$$

$$[(T_1 \; T_2)] \rightsquigarrow ([T_1] \; [T_2])$$

$$[\sigma] \rightsquigarrow \sigma$$

Diamond / Church-Rosser

$$
\begin{tikzcd}
& T_0 \arrow[dl, squiggly, "*"'] \arrow[dr, squiggly, "*"] & \\
T_1 \arrow[dr, squiggly, "*"'] & & T_2 \arrow[dl, squiggly, "*"] \\
& T_3 &
\end{tikzcd}
$$

=> Reduction order doesn't matter $^{(*)}$

Normal Form / End of Evaluation

$$
\begin{aligned}
T &:= \lambda \sigma . T \\
&\mid (T\ T) \\
&\mid \sigma
\end{aligned}
$$

$$\underbrace{(\lambda \sigma . T\ T)}_{\text{Redex}}$$

!not all terms have a normal form!

Shadowing / Scoping

$$\lambda x.(x\ \lambda x.x)$$

$$=$$

$$\lambda x.(x\ \lambda y.y)$$

Turing Complete

Programming!!

Definition / Constant

$$[0\text{-}9\,A\text{-}Z]^+ = T$$

Definition / Constant

$$[0 - 9 \; A - Z]^{+} = T$$

! First subst. Definitions, then evaluate !

Data Types & Structures

Booleans

$$TRUE = \lambda t.\lambda f.t$$

$$FALSE = \lambda t.\lambda f.f$$

$$OR = \lambda x.(x\ x)$$

$$AND = \lambda x.\lambda y.(x\ y\ x)$$

Boolean Example

(OR FALSE TRUE)

$$== (\underbrace{\lambda x.(x\ x)}_{\text{OR}}\ \underbrace{\lambda t.\lambda f.f}_{\text{FALSE}}\ \underbrace{\lambda t.\lambda f.t}_{\text{TRUE}})$$

Boolean Example

$$(\lambda x.(x\ x)\ \ \lambda t.\lambda f.f\ \ \lambda t.\lambda f.t)$$

$$\rightsquigarrow (\lambda t.\lambda f.f\ \ \lambda t.\lambda f.f\ \ \lambda t.\lambda f.t)$$

Boolean Example

$$(\lambda x.(x\ x)\ \lambda t.\lambda f.f\ \lambda t.\lambda f.t)$$

$$\rightsquigarrow (\underbrace{\lambda t.\lambda f.f\ \lambda t.\lambda f.f}\ \lambda t.\lambda f.t)$$

$$\rightsquigarrow (\lambda f.f\ \lambda t.\lambda f.t)$$

Boolean Example

$$(\lambda x.(x\ x)\ \lambda t.\lambda f.f\ \lambda t.\lambda f.t)$$

$$\rightsquigarrow (\underbrace{\lambda t.\lambda f.f\ \lambda t.\lambda f.f}_{\text{inception!}}\ \lambda t.\lambda f.t)$$

$$\rightsquigarrow (\lambda f.f\ \lambda t.\lambda f.t)$$

Boolean Example

$$(\lambda f.f\ \lambda t.\lambda f.t)$$

$$\rightsquigarrow \lambda t.\lambda f.t == \text{TRUE}$$

$\uparrow$ ok ✓

Booleans

$$TRUE = \lambda t.\lambda f.t$$

$$FALSE = \lambda t.\lambda f.f$$

$$NOT = ???$$

Booleans

$$TRUE = \lambda t.\lambda f.t$$

$$FALSE = \lambda t.\lambda f.f$$

$$NOT = \lambda x.(x\ FALSE\ TRUE)$$

"if"

Generic Encodings

~ Normalform

~ Selector

data

$$\lambda\sigma.(\underbrace{\sigma}_{\text{selector}}\ T_1\ T_2 \cdots)$$

Church Pair

constructor

$$PAIR = \lambda a.\lambda b.\lambda x.(x\ a\ b)$$

$$(PAIR\ A\ B) == \lambda x.(x\ A\ B)$$

data

selector

Church Pair

$$\lambda x.(x\ A\ B)$$

$$FST = \lambda a.\lambda b.a$$
$$SND = \lambda a.\lambda b.b$$

selector

Church Pair

$$\lambda x.(x\ A\ B)$$

$$FST = \lambda a.\lambda b.a$$
$$SND = \lambda a.\lambda b.b$$

selector

$$\begin{bmatrix} == TRUE \\ == FALSE \end{bmatrix}$$

Church Pair

$$SND = \lambda a . \lambda b . b$$

$$(\lambda x . (x\ A\ B)\ SND)$$
$$\rightsquigarrow (\lambda a . \lambda b . b\ A\ B)$$
$$\rightsquigarrow (\lambda b . b\ B)$$
$$\rightsquigarrow B\ \checkmark$$

Church Pair

$$SND' = \lambda a.\lambda b.b$$

$$(SND\ \lambda x.(x\ A\ B))\ ??$$

Church Pair

$$SND' = \lambda a . \lambda b . b$$

$$(SND\ \lambda x.(x\ A\ B))$$

$$\Rightarrow SND = \lambda r.(r\ SND')$$

Church List

$$\lambda a.(a\ A\ \lambda b.(b\ B\ \lambda c.(c\ C\ \underbrace{\lambda x.\lambda y.y}_{\text{NIL/FALSE}})))$$

Church Numerals

$$0 = \lambda s.\lambda z.\ z$$

$$1 = \lambda s.\lambda z.(s\ z)$$

$$2 = \lambda s.\lambda z.(s\ (s\ z))$$

$$3 = \lambda s.\lambda z.(s\ (s\ (s\ z)))$$

$$\vdots$$

Church Numerals

$$3 = \lambda s.\lambda z.(s\ (s\ (s\ z)))$$

EVEN?

TRUE

FALSE

Church Numerals

$$(\lambda s.\lambda z.(s\ (s\ (s\ z)))\ \mathrm{NOT}\ \mathrm{TRUE})$$

$$\rightsquigarrow \mathrm{NOT}\ (\mathrm{NOT}\ (\mathrm{NOT}\ \mathrm{TRUE}))$$

$$\rightsquigarrow \mathrm{FALSE}$$

Church Numerals

$$(\lambda x.(x\ x)\ \ \lambda s.\lambda z.(s\ (s\ z)))$$

Church Numerals

$$(\lambda x.(x\ x)\ \ \lambda s.\lambda z.(s\ (s\ z)))$$

$$\rightsquigarrow (\lambda s.\lambda z.(s\ (s\ z))\ \lambda s.\lambda z.(s\ (s\ z)))$$

Church Numerals

$$(\lambda x.(x\ x)\ \ \lambda s.\lambda z.(s\,(s\ z)))$$

$$\rightsquigarrow (\lambda s.\lambda z.(s\,(s\ z))\ \ \lambda s.\lambda z.(s\,(s\ z)))$$

$$\rightsquigarrow \lambda z.(\lambda s.\lambda z.(s\,(s\ z))\,(\lambda s.\lambda z.(s\,(s\ z))\ z))$$

Church Numerals

$$(\lambda x.(x\ x)\ \ \lambda s.\lambda z.(s\ (s\ z)))$$

$$\rightsquigarrow (\lambda s.\lambda z.(s\ (s\ z))\ \lambda s.\lambda z.(s\ (s\ z)))$$

$$\rightsquigarrow \lambda z.(\lambda s.\lambda z.(s\ (s\ z))\ (\lambda s.\lambda z.(s\ (s\ z))\ z))$$

$$\vdots$$

$$\rightsquigarrow \lambda s.\lambda z.(s\ (s\ (s\ (s\ z))))$$

$$\Rightarrow n^n!$$

Church Numeral Functions

$$\text{SUCC} = \lambda n.\lambda s.\lambda z.(s\ (n\ s\ z))$$
$$\text{ADD} = \lambda a.\lambda b.\lambda f.\lambda x.(a\ f\ (b\ f\ x))$$
$$\text{MUL} = \lambda a.\lambda b.\lambda f.(a\ (b\ f))$$
$$\text{POW} = \lambda a.\lambda b.(b\ a)$$

Church Numeral Functions

$$SUCC = \lambda n.\lambda s.\lambda z.(s\ (n\ s\ z))$$

$$ADD = \lambda a.\lambda b.\lambda f.\lambda x.(a\ f\ (b\ f\ x))$$

$$MUL = \lambda a.\lambda b.\lambda f.(a\ (b\ f))$$

$$POW = \lambda a.\lambda b.(b\ a)$$

$$PRED = \lambda n.\lambda f.\lambda x.(n\ \lambda g.\lambda h.(h\ (g\ f))\ \lambda \sigma.x\ \lambda u.u)$$

$$SUB = \lambda a.\lambda b.(b\ PRED\ a)$$

$f\ x))$

$\text{PRED} = \lambda n.\lambda f.\lambda x.(n\ \lambda g.\lambda h.(h\ (g\ f))\ \lambda \sigma.x\ \lambda u.u)$

$\text{SUB} = \lambda a.\lambda b.(b\ \text{PRED}\ a)$

Other Numeral Encodings

unary: $\lambda s.\lambda z.(s\,(s\,(\cdot\cdot(s\ z)\cdot\cdot)))$

binary: $\lambda t.\lambda f.\lambda z.(t\,(f\,(f\,(t\ z))))$ $= 1001_2 = 9$

n-ary: $\lambda b_n.\ \cdots \lambda b_0.\lambda z.(\cdots(\cdots\ z)\cdots)$

⋮

Other Data

- Char = Number
- String = List of Chars
- Tree = List of Lists / Tuples (e.g. AVL)
- Hashmap = Tree
- Structs & Enums = Custom with selectors
- Rational/Real/Complex = Pairs of Numbers (see bruijn)
- I/O: List of Chars

Recursion

Factorial

$x! = x \cdot (x-1) \cdot (x-2) \cdots 1$

$$\text{FAC} = \lambda x.(\underbrace{\text{ZERO? } x}_{\text{"if"}}\ 1\ (\text{MUL } x\ (\text{FAC } (\text{PRED } x))))$$

Factorial

$$
\begin{aligned}
\mathrm{FAC} &= \lambda x.(\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ x\ (\mathrm{FAC}\ (\mathrm{PRED}\ x))))\\
&= (\lambda f.\ \lambda x.(\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ x\ (f\ (\mathrm{PRED}\ x))))\ \mathrm{FAC})
\end{aligned}
$$

Factorial

$$
\begin{aligned}
\mathrm{FAC} &= \lambda x.(\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ x\ (\mathrm{FAC}\ (\mathrm{PRED}\ x)))) \\
&= (\lambda f.\lambda x.(\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ x\ (f\ (\mathrm{PRED}\ x))))\ \mathrm{FAC}) \\
&= (\lambda f.\lambda x.(\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ x\ (f\ (\mathrm{PRED}\ x)))) \\
&\quad\ \lambda f.\lambda x.(\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ x\ (f\ (\mathrm{PRED}\ x)))))
\end{aligned}
$$

Factorial

$$\mathrm{FAC} = \lambda x.(\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ x\ (\mathrm{FAC}\ (\mathrm{PRED}\ x))))$$

$$= (\lambda f.\ \lambda x.(\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ x\ (f\ (\mathrm{PRED}\ x))))\ \mathrm{FAC})$$

$$= (\lambda f.\ \lambda x.(\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ x\ (f\ (\mathrm{PRED}\ x))))$$

$$\lambda f.\ \lambda x.(\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ x\ (f\ (\mathrm{PRED}\ x)))))$$

$$= (\lambda f.\ \lambda x.(\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ x\ (f\ f\ (\mathrm{PRED}\ x))))$$

$$\lambda f.\ \lambda x.(\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ x\ (f\ f\ (\mathrm{PRED}\ x)))))\quad ✓$$

Better Factorial

$$\mathrm{FAC} = (\lambda f.\lambda x.(\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ \ x\ (f\ f\ (\mathrm{PRED}\ x))))$$
$$\lambda f.\lambda x.(\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ \ x\ (f\ f\ (\mathrm{PRED}\ x)))))$$

Better Factorial

$$\mathrm{FAC} = (\lambda f.\ \lambda x.\ (\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ x\ (f\ f\ (\mathrm{PRED}\ x))))$$

$$\lambda f.\ \lambda x.\ (\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ x\ (f\ f\ (\mathrm{PRED}\ x)))))$$

$$= (\lambda x.\ (x\ x))$$

$$\lambda f.\ \lambda x.\ (\mathrm{ZERO?}\ x\ 1\ (\mathrm{MUL}\ x\ (f\ f\ (\mathrm{PRED}\ x)))))$$

Better Factorial

$$\text{FAC} = (\lambda f.\lambda x.(\text{ZERO?}\ x\ 1\ (\text{MUL}\ x\ (f\ f\ (\text{PRED}\ x))))$$

$$\lambda f.\lambda x.(\text{ZERO?}\ x\ 1\ (\text{MUL}\ x\ (f\ f\ (\text{PRED}\ x)))))$$

$$= (\lambda x.(x\ x)$$

$$\lambda f.\lambda x.(\text{ZERO?}\ x\ 1\ (\text{MUL}\ x\ (f\ f\ (\text{PRED}\ x)))))$$

$$= (\lambda f.(\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x)))$$

$$\lambda f.\lambda x.(\text{ZERO?}\ x\ 1\ (\text{MUL}\ x\ (f\ (\text{PRED}\ x)))))$$

Fixpoint

$$\Theta = (\lambda f.\lambda x.(x\ (f\ f\ x))$$
$$\qquad \lambda f.\lambda x.(x\ (f\ f\ x)))$$

$$Y = \lambda f.(\lambda x.f(x\ x)\ \lambda x.f(x\ x))$$

$$Z = \lambda f.(\lambda x.(f\ \lambda y.(x\ x\ y))$$
$$\qquad \lambda x.(f\ \lambda y.(x\ x\ y)))$$

$$\vdots$$

Relaxation: Drawing Images

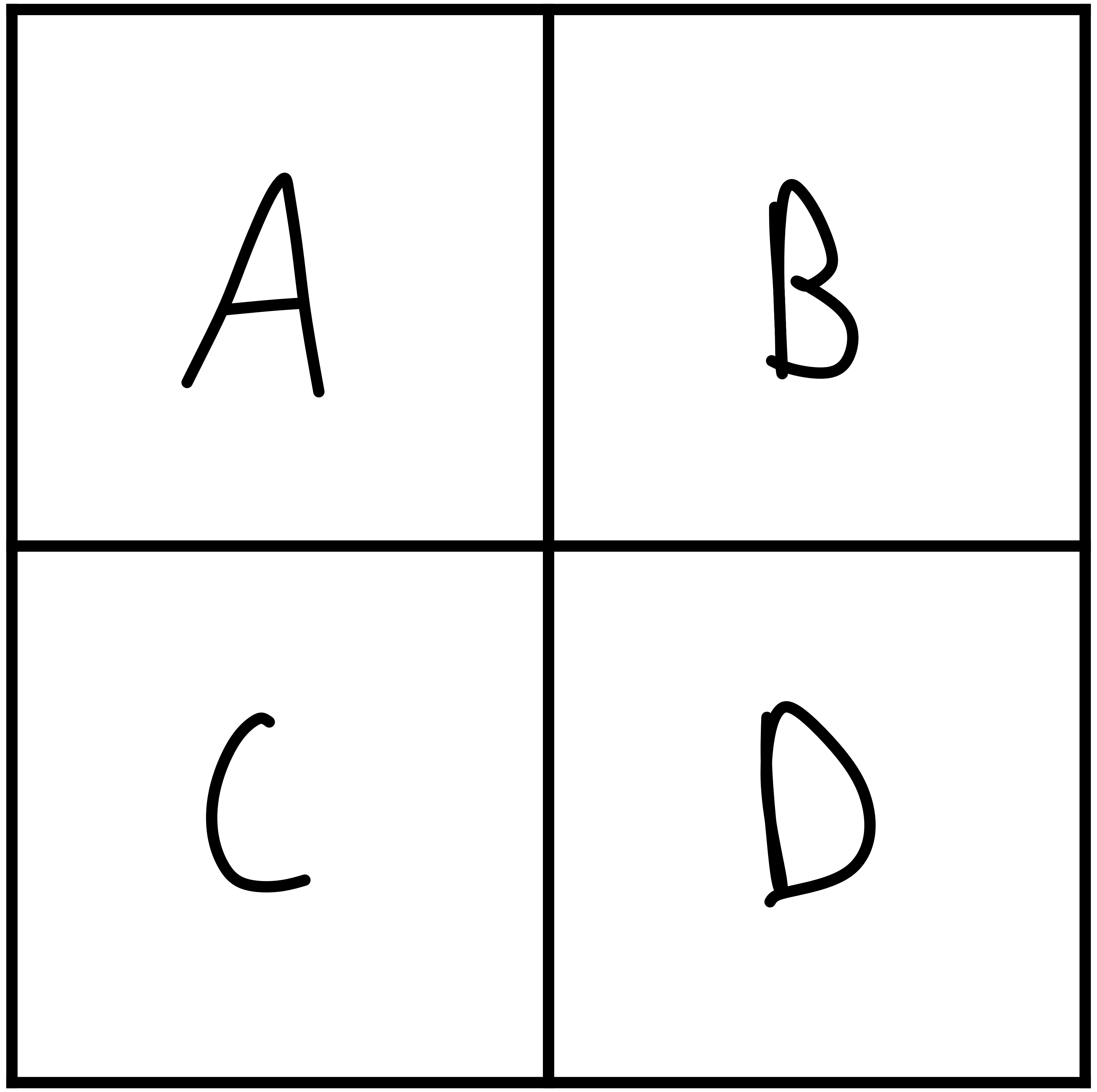
Lambda Screen

$$\text{pixel} := \{1, 0\}$$

$$\text{screen} := \lambda x.(x\ \underbrace{A\ B\ C\ D}_{\rightsquigarrow \{\text{pixel}, \text{screen}\}})$$

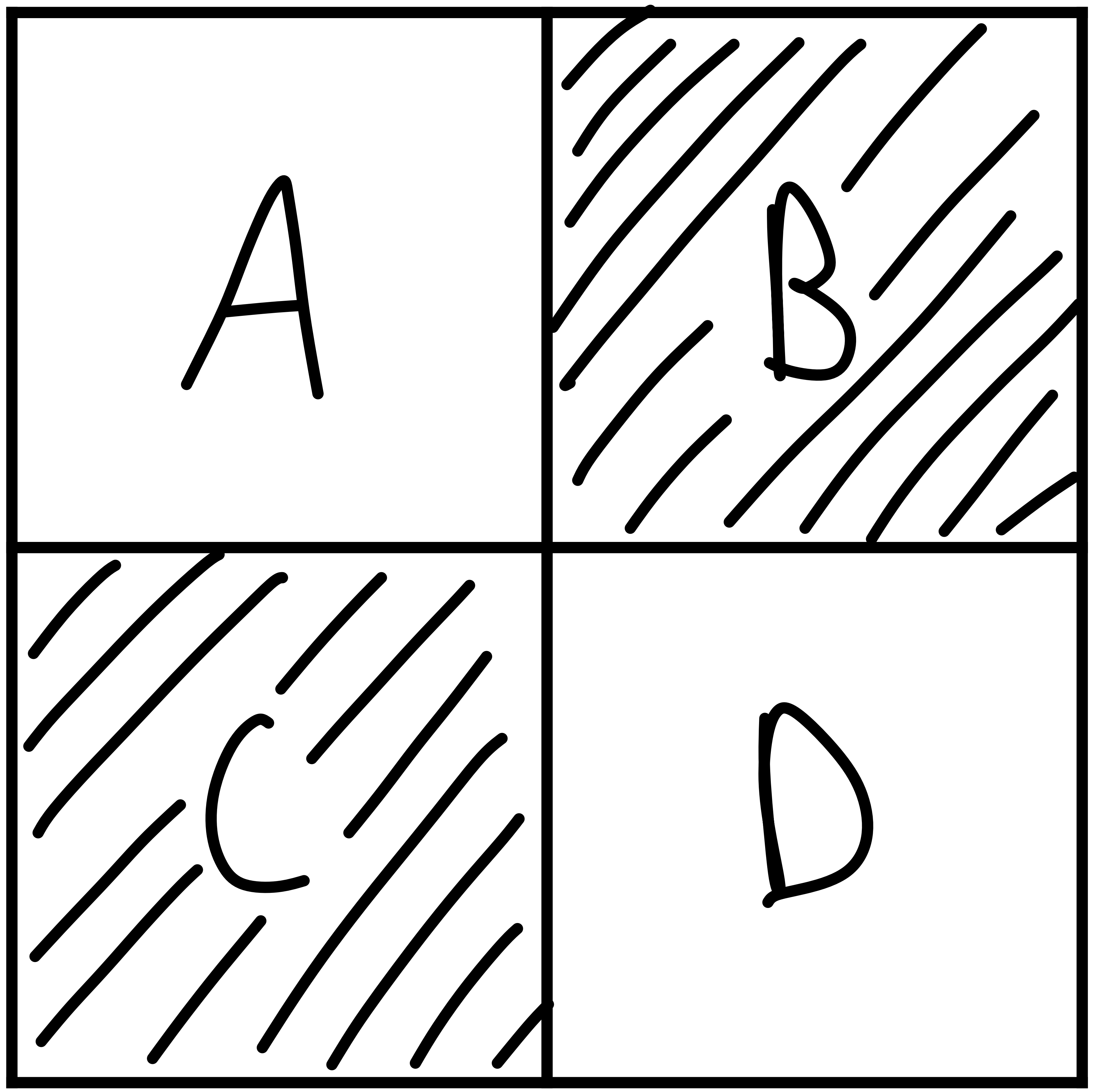
Lambda Screen

$\lambda x.(x\ A\ B\ C\ D)$



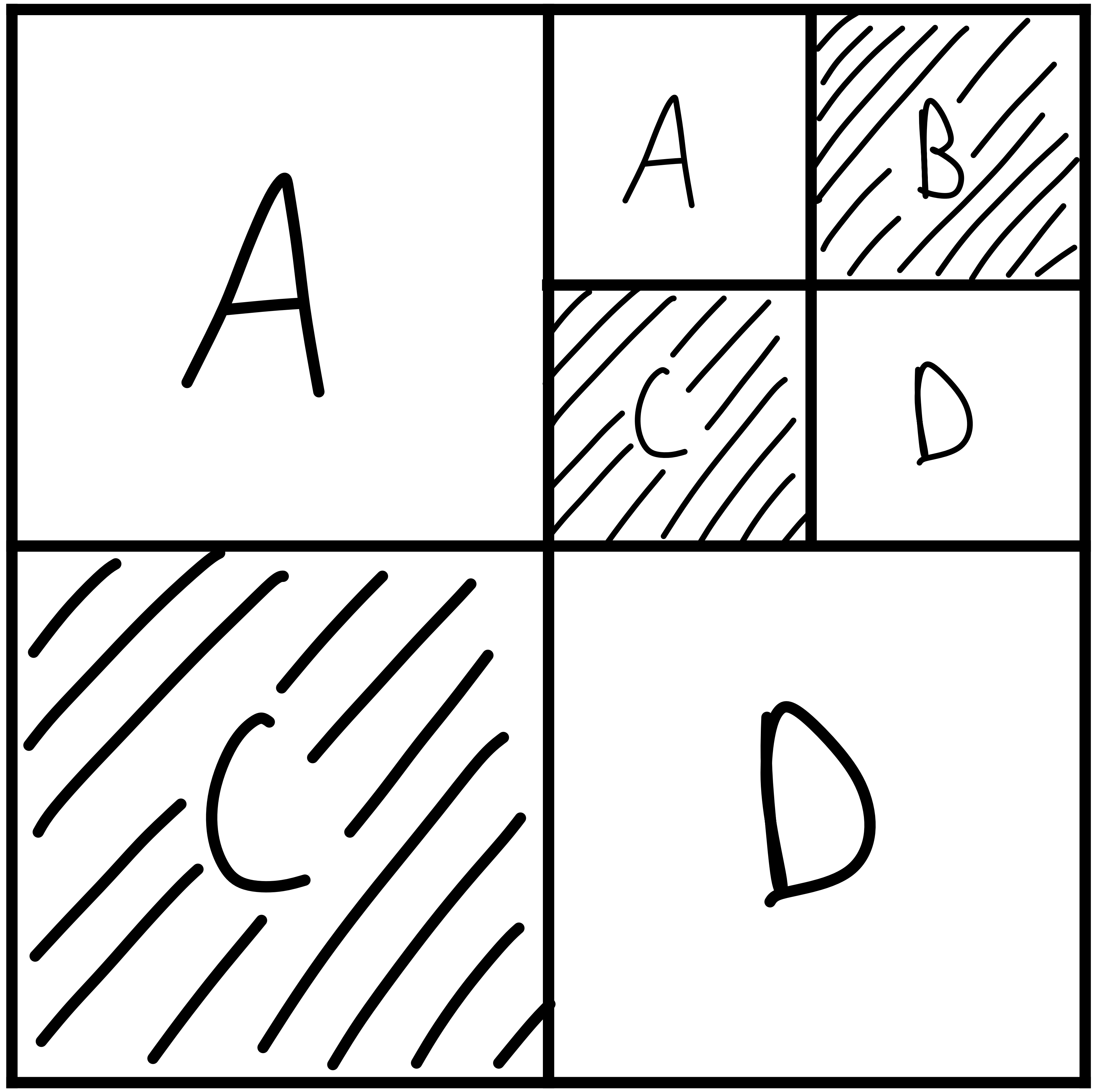
Lambda Screen

$\lambda x.(x\ 1\ 0\ 0\ 1)$



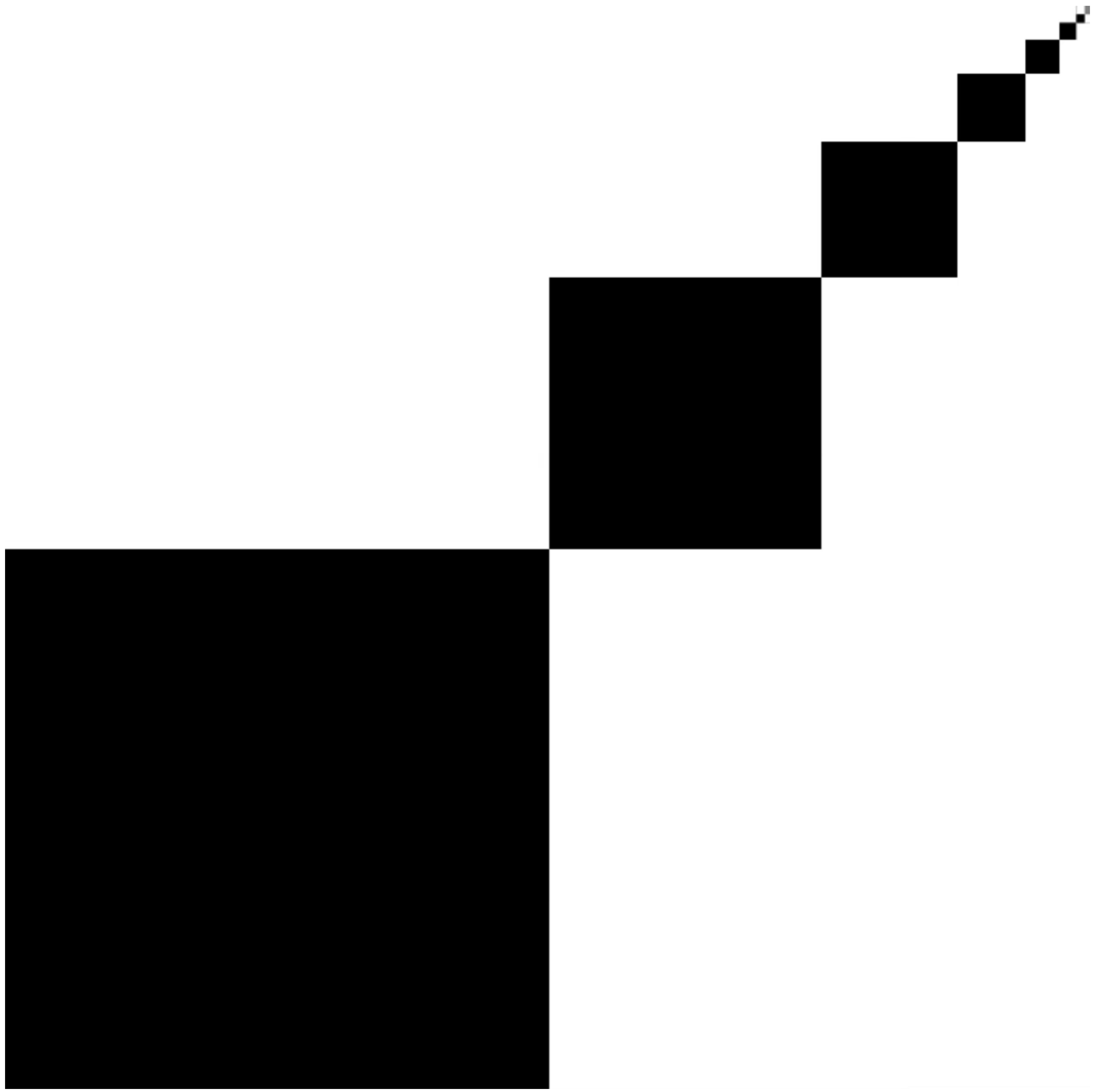
Lambda Screen

$$\lambda x_0.(x_0\ 1\ \lambda x_1.(x_1\ 1\ 0\ 0\ 1)\ 0\ 1)$$



Lambda Screen

$$(Y\ \lambda f.\ \lambda x_0.(x_0\ 1\ f\ 0\ 1))$$



Lambda Screen

$(Y\ \lambda f.$

$\quad \lambda x_0.(x_0$

$\quad\quad f$

$\quad\quad 1$

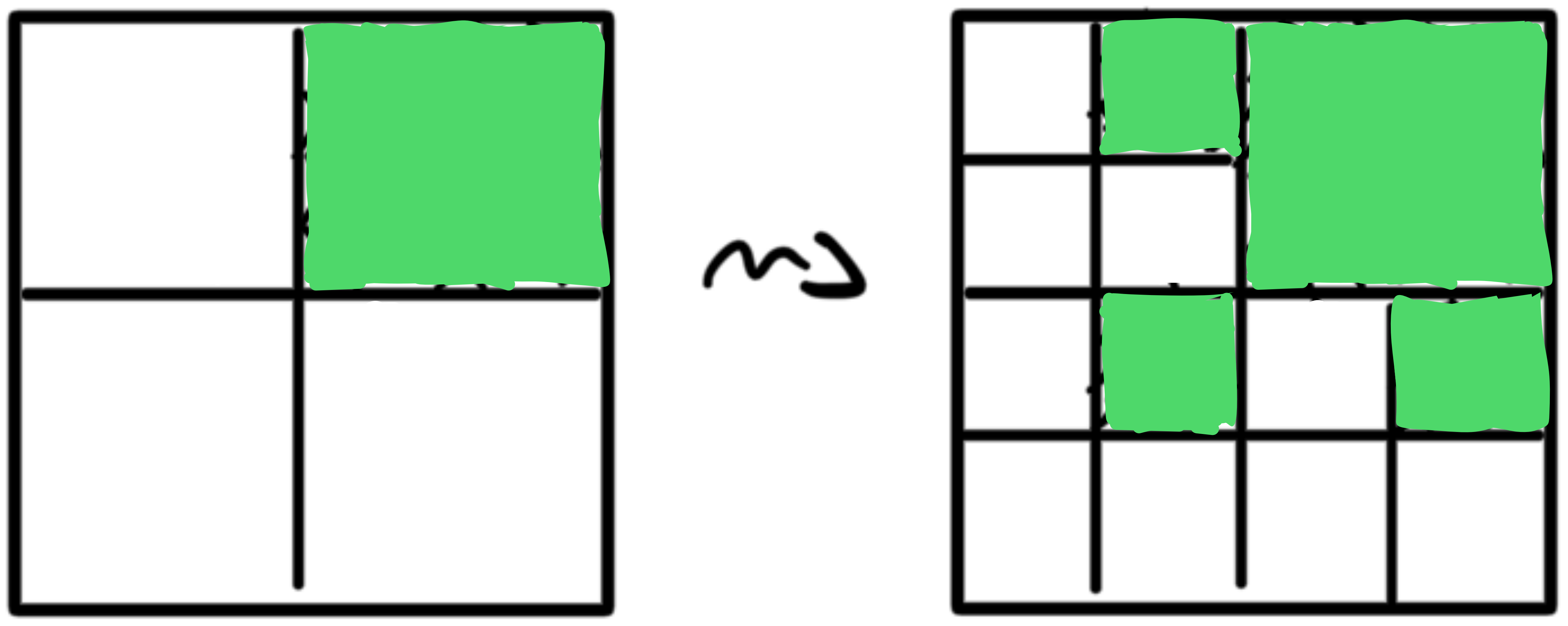
$\quad\quad f$

$\quad\quad f))$

???

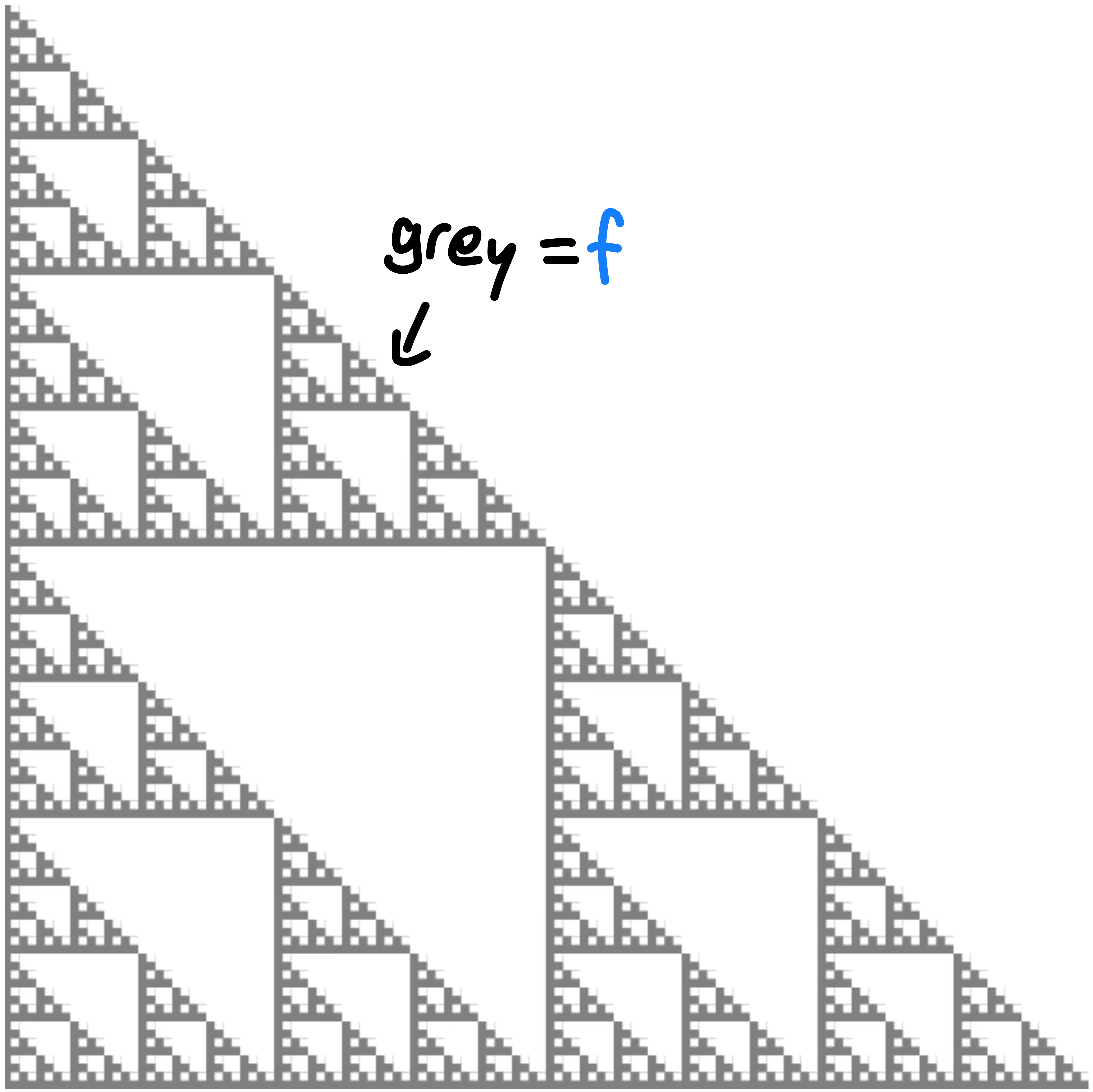
Lambda Screen

$$(Y\ \lambda f.\ \lambda x_0.(x_0\ f\ 1\ f\ f))$$



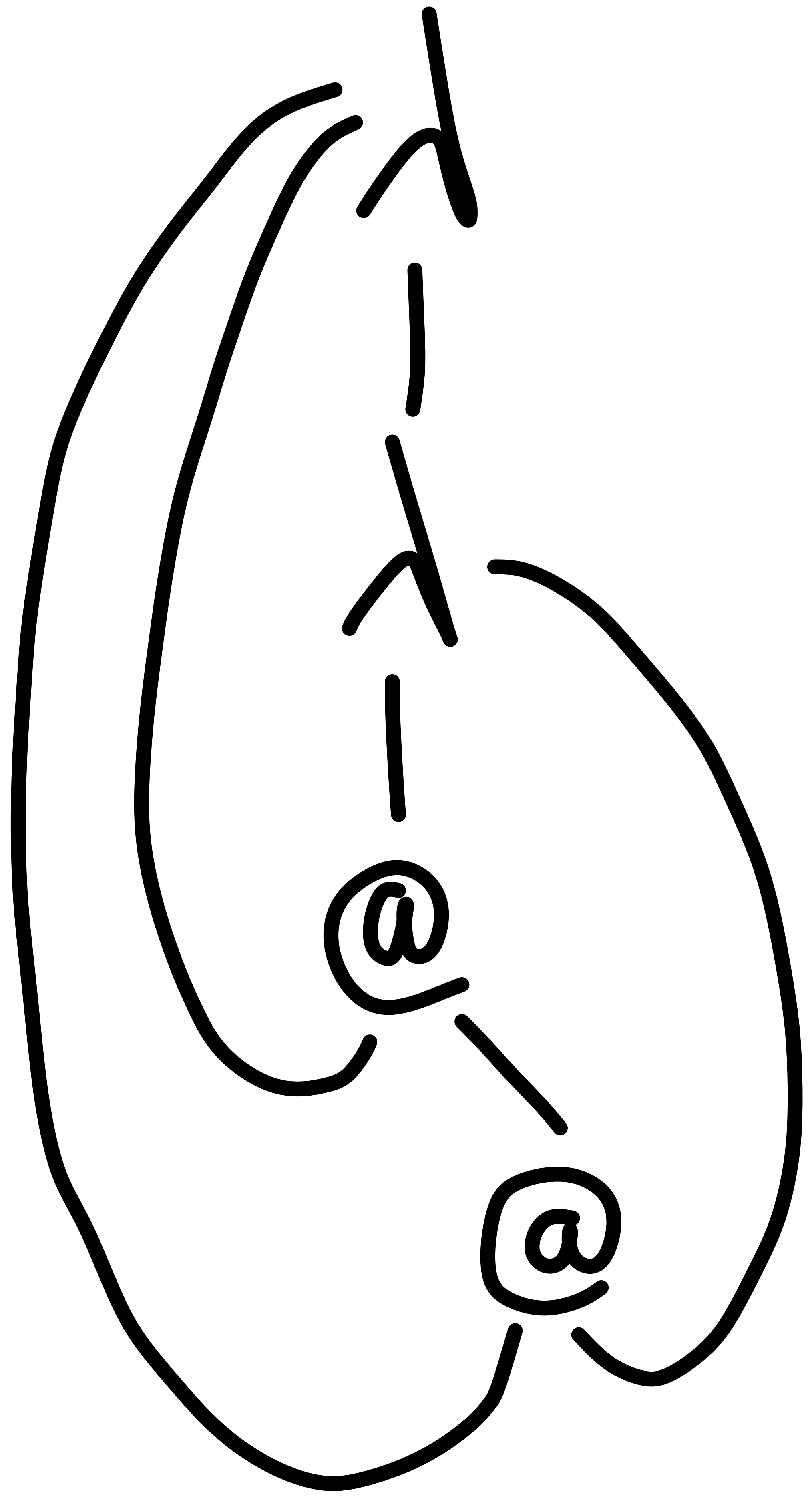
Lambda Screen

$$(Y\ \lambda f.\ \lambda x_0.(x_0\ f\ \neg\ f\ f))$$



Lambda Graph

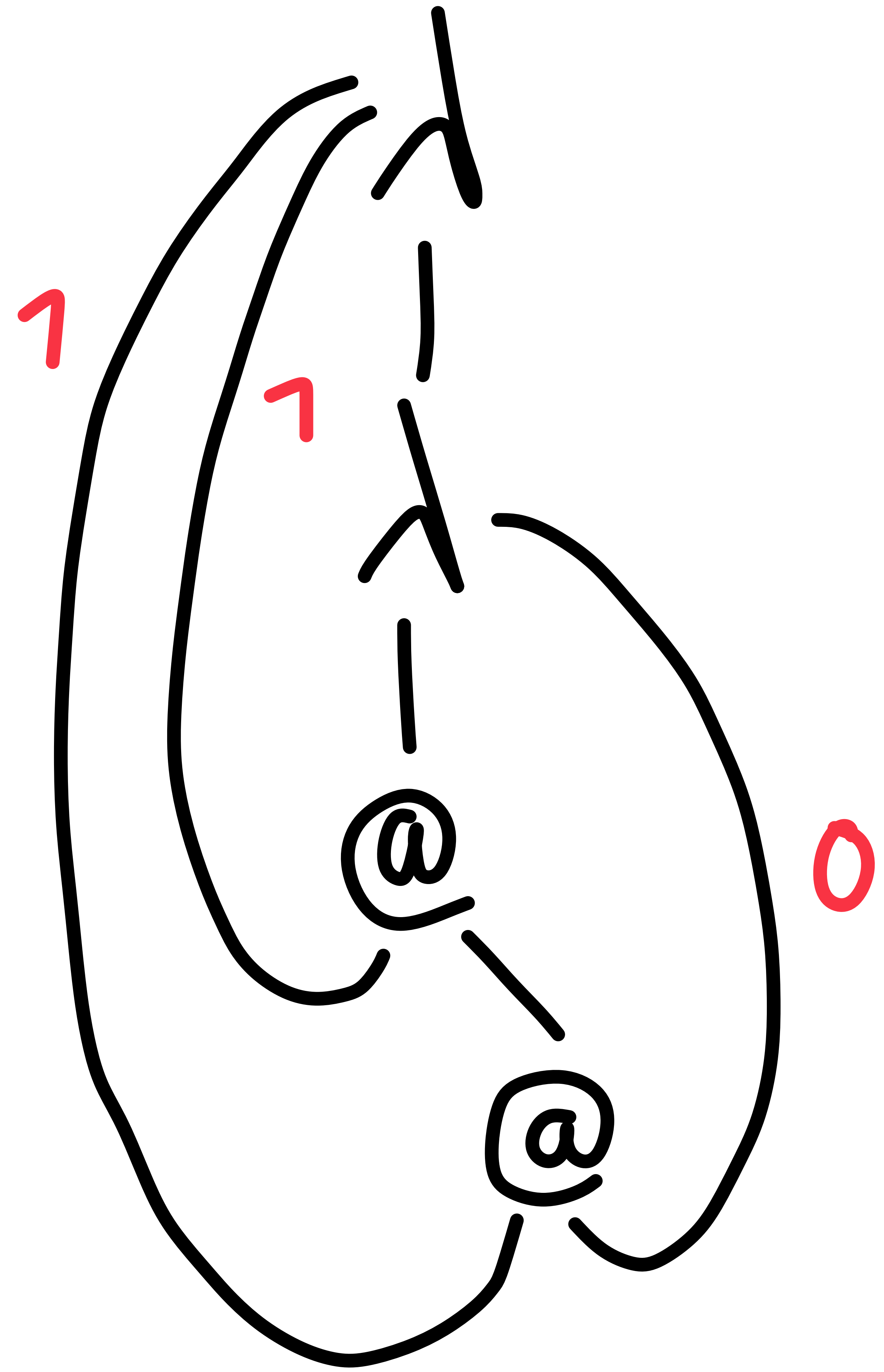
$\lambda a. \lambda b. (a\ (a\ b)) ==$



de Bruijn Indices

$$\lambda a.\lambda b.(a\ (a\ b)) ==$$

$$== \lambda\lambda(1\ (1\ 0))$$



Combinators

~ normal form
~ short

$$S = \lambda\lambda\lambda(2\ 0\ (1\ 0))$$
$$K = \lambda\lambda 1$$
$$I = \lambda 0$$
$$B = \lambda\lambda\lambda(2\ (1\ 0))$$
$$\vdots$$

infinite-apply

bruijn

- Syntax for pure lambda calculus
- Standard library (730+ definitions)
- No primitive functions

$\lambda x.x$

- DECT := 2222
- mastodon := @marvin@types.pl
- website := marvinborner.de
- links := {bruijn, text, lambda-screen, infinite-apply}.<website>
- slides := github.com/marvinborner/gpn22