λ-calculus & categories 7

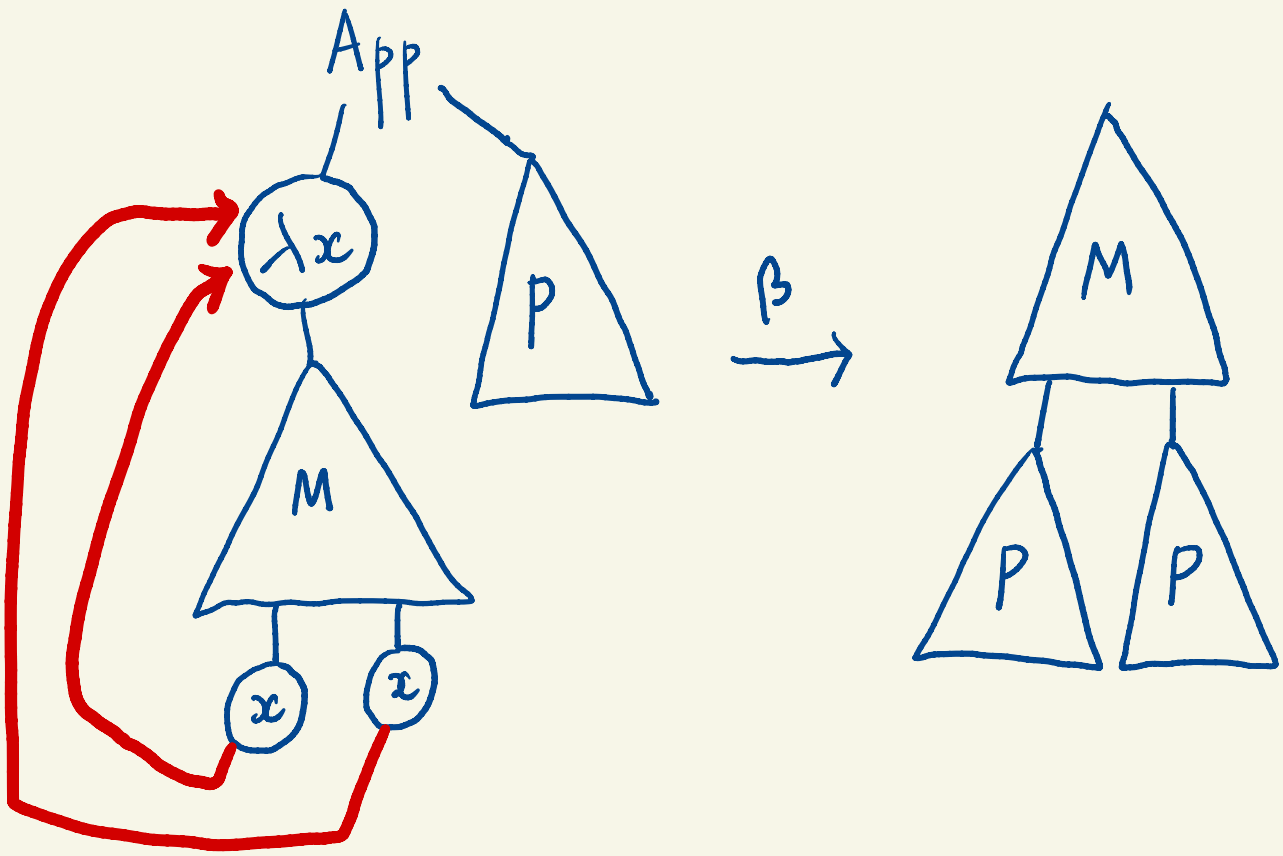
Monday 16 November 2020

Computing with λ-terms

Computing with λ-terms.

β-rule:

$$App(\lambda x.M, P) \longrightarrow M[x := P]$$



the symbolic/syntactic description of function application.

the η-rule

the variable x is not free in M.

$$M \longrightarrow \lambda x.\, App(M, x)$$

"every λ-term M is a function"

the function which takes an argument P and returns $App(M, P)$

here formulated as an "η-expansion" instead of an "η-reduction".

A remark.

given two λ-terms M and N

we define the composite of M and N

$$(M \circ N)$$

as the λ-term

$$\lambda x.\ App(M, App(N, x))$$

$App(N, x)$: we apply N to x

$App(M, App(N, x))$: we apply M to the result

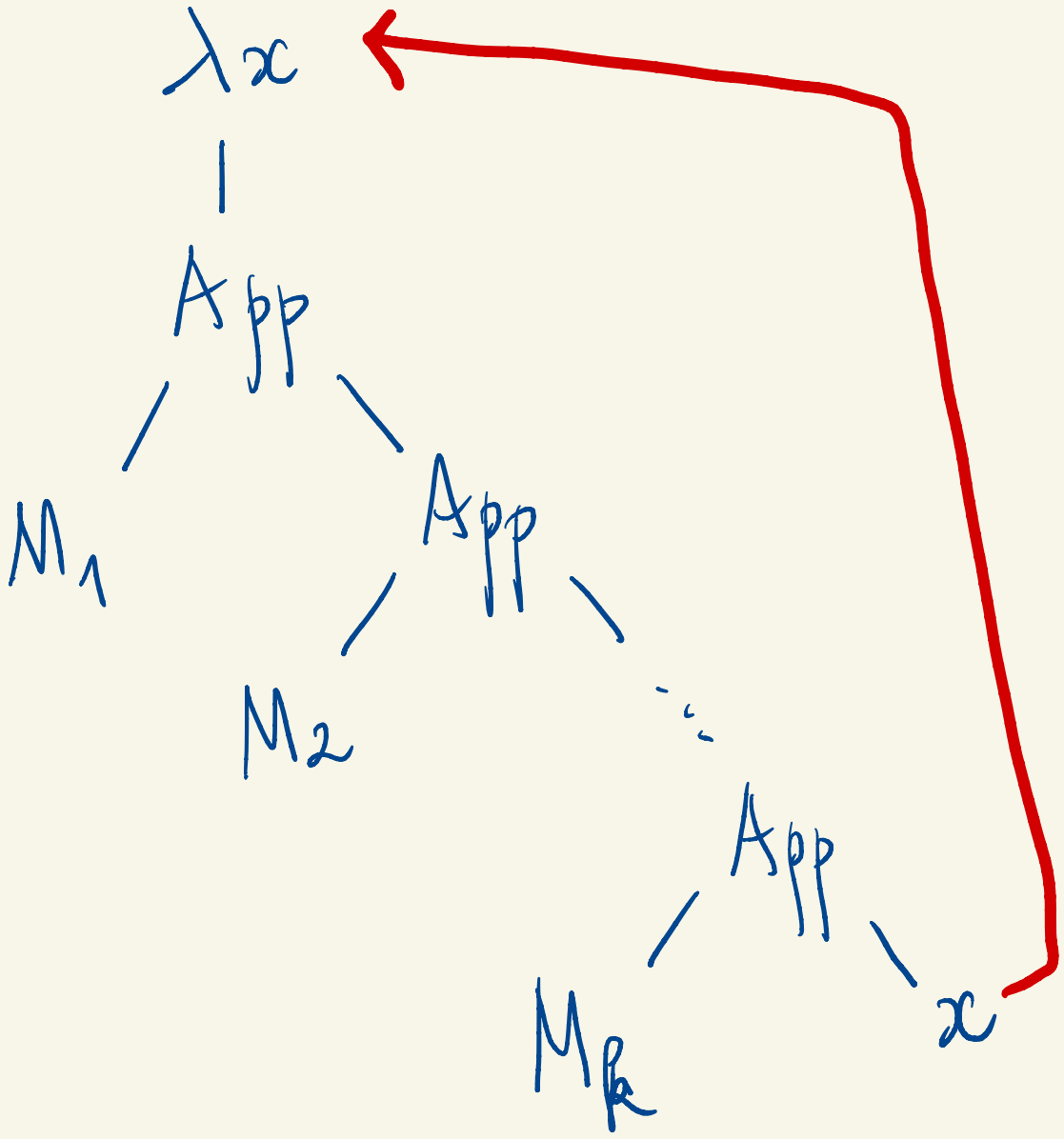
this can be generalised to any sequence $M_1, \ldots, M_k$ of λ-terms

$$(M_1 \circ \cdots \circ M_k)$$

is defined as

$$\lambda x.\, App(M_1, App(M_2, \ldots App(M_k, x))\ldots)$$

where x is "fresh", ie that it is not free in $M_1, \ldots, M_k$



Given three λ-terms M_1, M_2, M_3

$$(M_1 \circ M_2) \circ M_3$$

$$= \lambda x.\ App\big((M_1 \circ M_2), App(M_3, x)\big)$$

$$= \lambda x.\ App\Big(\big(\lambda y.\ App(M_1, App(M_2, y))\big), App(M_3, x)\Big)$$

Function — argument

$$\xrightarrow{\beta} \lambda x.\ App\big(M_1, App(M_2, App(M_3, x))\big)$$

obtained by replacing y by $App(M_3, x)$ in the λ-term $App(M_1, App(M_2, y))$

$$= M_1 \circ M_2 \circ M_3$$

<<λax associative>>

$$M_1 \circ (M_2 \circ M_3)$$

$$= \lambda x.\ App(M_1, App(M_2 \circ M_3, x))$$

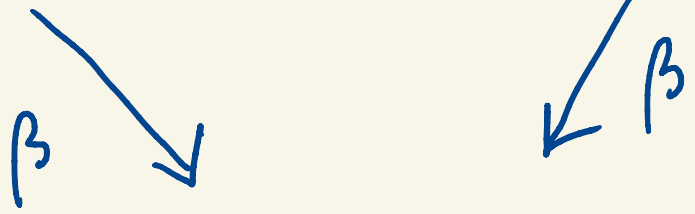
$$= \lambda x.\ App(M_1, App(\underbrace{\lambda y.\ App(M_2, App(M_3, y))}_{M_2 \circ M_3}, x))$$

$$\xrightarrow{\beta} \lambda x.\ App(M_1, App(M_2, App(M_3, x)))$$

$$= M_1 \circ M_2 \circ M_3$$

to summarize :

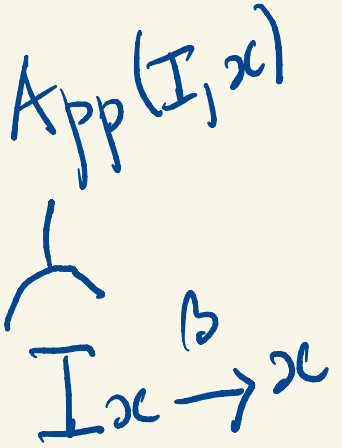
$(M_1 \circ M_2) \circ M_3$ $\qquad$ $M_1 \circ (M_2 \circ M_3)$



$M_1 \circ M_2 \circ M_3$

What about identities ?

$I = \lambda x . x$



$M \circ I = \lambda x . App(M, App(I, x))$

$\xrightarrow{\beta} \lambda x . App(M, x)$

$M \circ I$ is equivalent to M modulo β, η

$$I \circ M = \lambda x.\ App\,(I,\ App\,(M, x))$$

$$\xrightarrow{\beta} \lambda x.\ App\,(M, x)$$

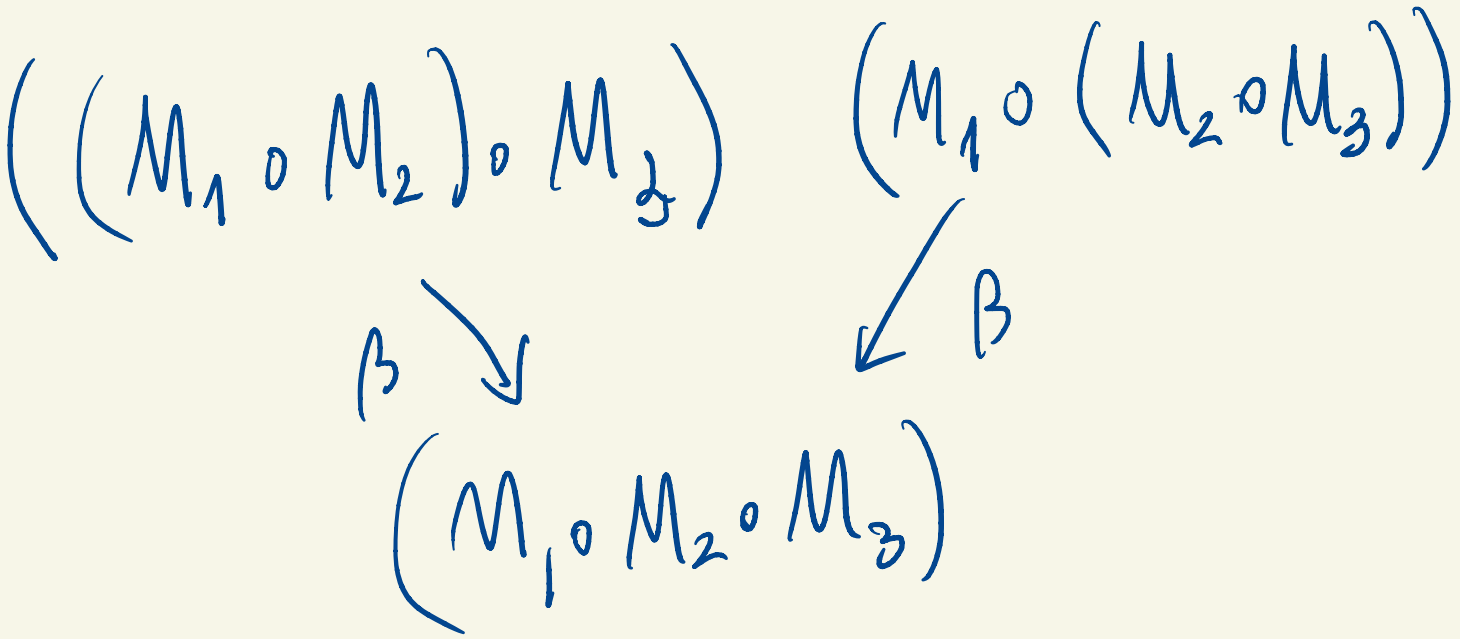
$I \circ M$ is equivalent to M modulo β, η

A natural idea:

$$(M) = \lambda x.\ App\,(M, x)$$

we can think of this operation as a unary form of composition

$$M \xrightarrow{\eta} (M)$$



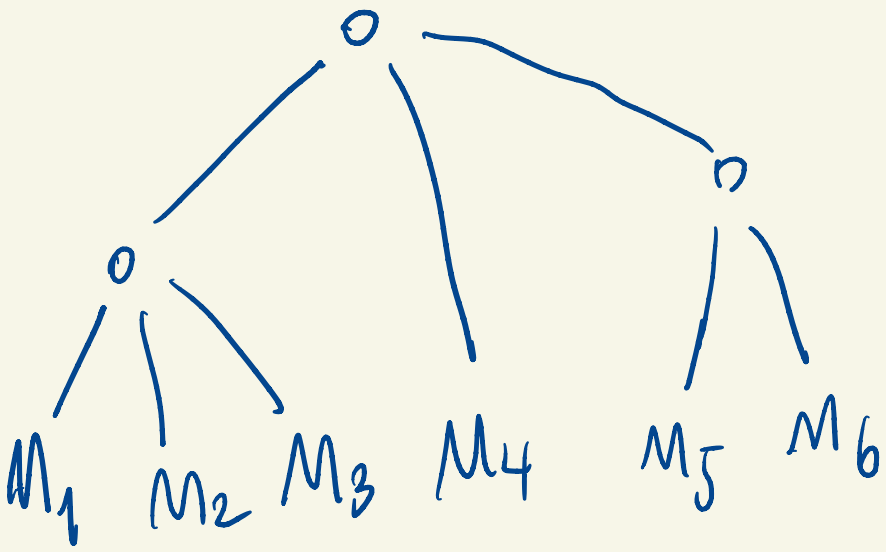
this defines something called
a "lax category"
just like a category
except that needs to consider
a family of n-ary compositions
with associativity and neutrality
"oriented" (in that case by $\beta\eta$)

Exercise: show that any λ-term obtained by composing a sequence

$$M_1, \ldots, M_k$$

of λ-terms in an arbitrary way

← "bracketed"



$$\big((M_1 \circ M_2 \circ M_3) \circ M_4 \circ (M_5 \circ M_6)\big)$$

rewrites to $(M_1 \circ \ldots \circ M_k)$

Let us move to booleans

and natural numbers.

the idea by **Alonzo Church** is that

the constants true and false

can be interpreted as the λ-terms:

$$True = \lambda x.\lambda y.\, x$$ first projection

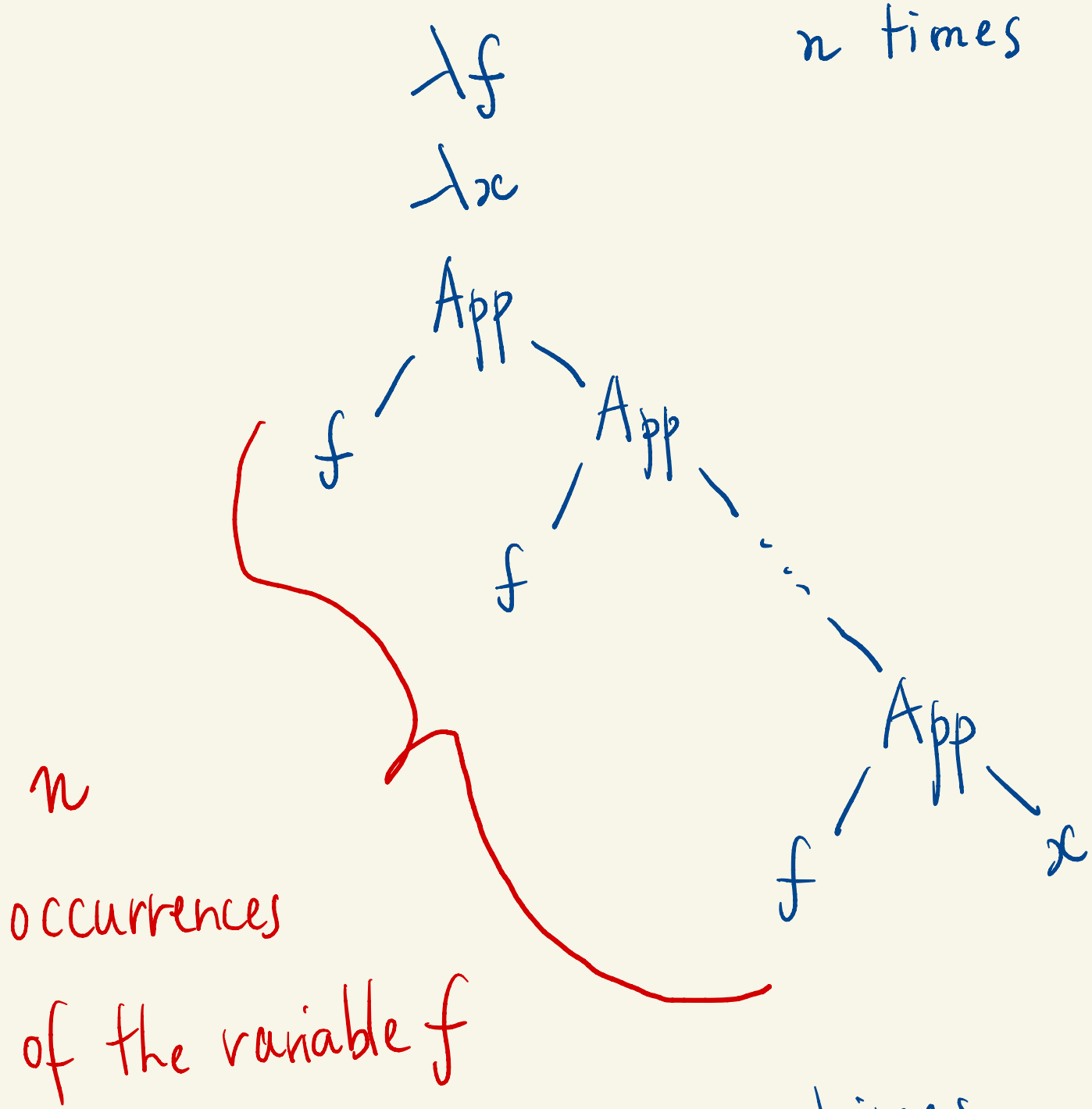
$$False = \lambda x.\lambda y.\, y$$ second projection

and every natural number (n)

can be interpreted as the λ-term:

$$[n] = \lambda f\, \lambda x.\, App(f, App(f, \ldots x))$$

$$= \lambda f.\ \lambda x.\ \underbrace{f\ f\ f\ f \ldots f}_{n \text{ times}}\ x$$



$$[n] = \lambda f.\ (\underbrace{f \circ f \circ \ldots \circ f}_{n \text{ times}})$$

So the natural number n

is interpreted as

the **functional**

which takes a function f

and returns f applied to itself

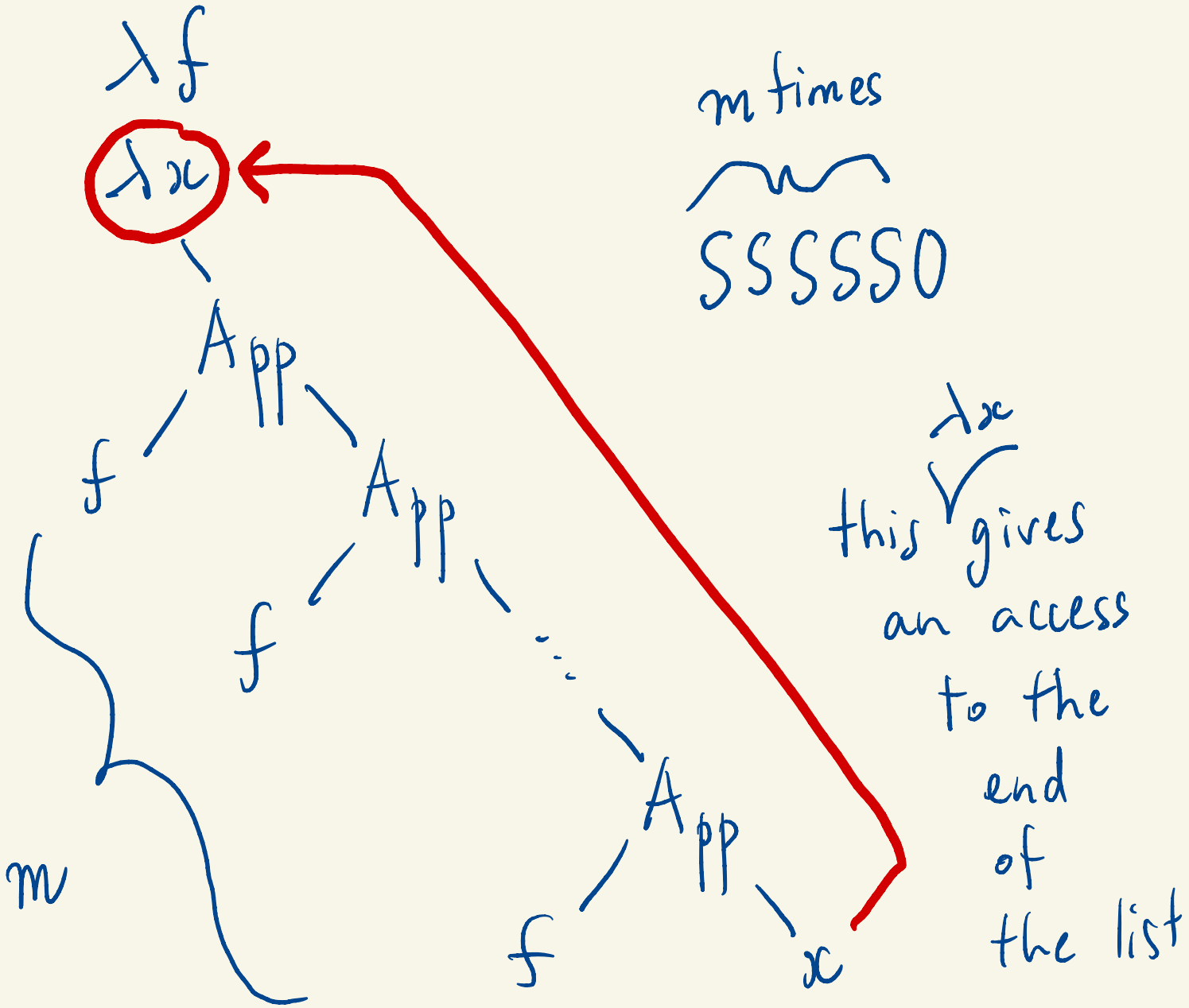
n times!

the Church encoding

of natural numbers.

Let us encode addition of two Church numerals:

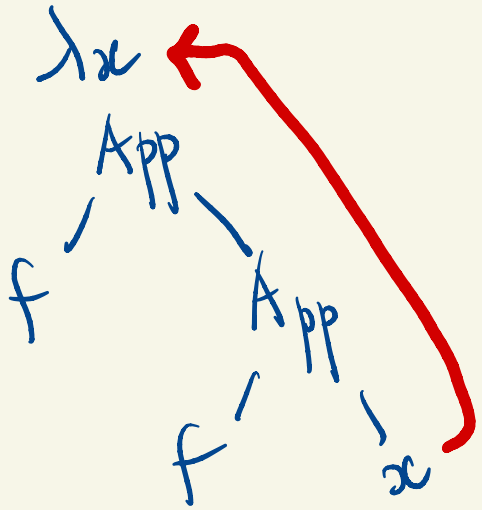
Add [m] [n] ⟶ [m+n]

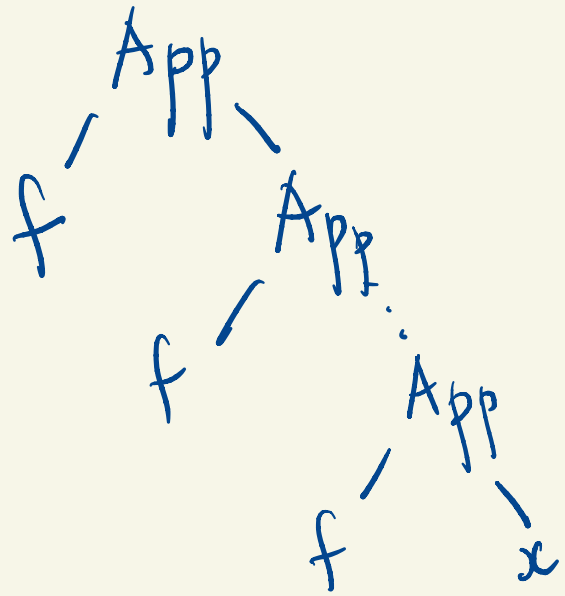


$$[m+n] = \boxed{\lambda f}.\ \lambda x \ldots$$

$$Add = \lambda m . \lambda n . \lambda f . \lambda x$$

$$App\left(\underbrace{App(m, f)}, \underbrace{App\left(\underbrace{App(n, f)}, x\right)}\right)$$





$$Add\ [m][n] \xrightarrow{\beta} [m+n]$$

$$App\left(App(Add, [m]), [n]\right)$$

$$Add = \lambda m.\lambda n.\lambda f.$$
$$\left(App(m,f) \circ App(n,f)\right)$$

$$f^{\circ m} \circ f^{\circ n} = \underbrace{(f \circ f \circ \cdots \circ f)}_{m} \circ \underbrace{(f \circ \cdots \circ f)}_{n}$$

$\downarrow$ β rules

$$f^{\circ m+n} = \underbrace{(f \circ \cdots \circ f)}_{m+n}$$

So we can add Church numerals.

moreover we can compose λ-terms.

what is missing in order to
encode every recursive function

$$f : \mathbb{N}^k \longrightarrow \mathbb{N}$$

as a λ-term $[f]$

such that for all sequence

$$m_1 \dots m_k \text{ of natural numbers}$$

$$[f]\,[m_1] \dots [m_k] \xrightarrow[\beta\eta]{} [n]$$

iff

$$f(m_1, \ldots, m_k) \simeq n$$

Thm. Every recursive function

$$f : \mathbb{N}^k \longrightarrow \mathbb{N}$$

can be encoded

by a λ-term $[f]$ in that way.

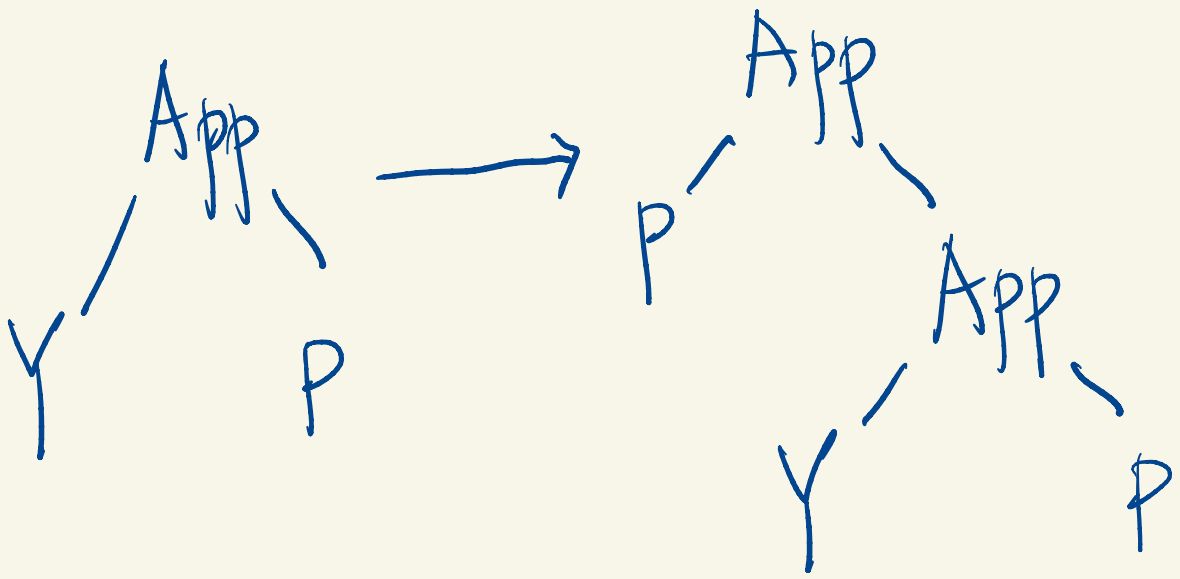
that means that the λ-calculus

has the computational power

of a Turing machine.

Essentially we need a fix point operator Y such that

$$Yp \longrightarrow p(Yp).$$



notice that the existence of such a fix point operator induces possibly non terminating compuations.

$$Yp \longrightarrow p(Yp) \longrightarrow pp(Yp) \longrightarrow \cdots$$

$$Y = \lambda f.\left(\lambda x.\, f(x\,x)\right)\left(\lambda x.\, f(x\,x)\right)$$

$$\underbrace{(\lambda x.\, x\,x)}_{\Delta}\,\underbrace{(\lambda x.\, x\,x)}_{\Delta}$$

$$\Delta\Delta \longrightarrow \Delta\Delta \longrightarrow \cdots$$

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

$$Y\,p = \lambda f.\left(\lambda x.\, f(x\,x)\right)\left(\lambda x.\, f(x\,x)\right)p$$

$$\xrightarrow{\beta} \left(\lambda x.\, p(x\,x)\right)\left(\lambda x.\, p(x\,x)\right)$$

$$\xrightarrow{\beta} p\left(\lambda x.\, p(x\,x)\right)\left(\lambda x.\, p(x\,x)\right)$$

$$\overset{\sim}{=} p\,(Y\,p)$$

that means that the pure calculus of function is sufficiently rich in order to accomodate "recursive calls".

I leave as exercise:

① the definition of ifzero

$$\text{ifzero } [0]\ P\ Q \to P$$

$$\text{ifzero } [n+1]\ P\ Q \to Q$$

② the definition of a recursive function using Y.

The simply-typed λ-calculus

the calculus of functions with types.

We start with a set of <u>type variables</u>

$TVar$ ← countable

A simple type is defined by the grammar

$$A ::= \alpha \in TVar \mid A \times B \mid A \Rightarrow B \mid \mathbb{1}$$

hence every type A is a tree constructed with $\times$, $\Rightarrow$, $\mathbb{1}$ and with leaves in $TVar$.

def. a context is a finite sequence

$$x_1 : A_1, \ldots, x_n : A_n$$

of pairs $x_i : A_i$ consisting of

- a term variable x_i
- a type A_i

We also make the assumption that all the variables x_i are different.

def: a typing judgement is a triple

$$x_1 : A_1, \ldots, x_n : A_n \vdash M : B$$

consisting of

- a context $\Gamma = x_1 : A_1, \dots, x_n : A_n$
- a λ-term M
- a type B

One could ask that all free var of M are in Γ

We want to show that a given λ-term M has a given type B in a given context Γ describing the types of the free variables of M.

To that purpose, we construct

derivation trees

establing the typing judgement

Def. a derivation tree is a tree whose branches are labelled by a typing judgement and whose nodes are labelled by the following typing rules:

6 rules

Logical rules

$$\text{var}\ \frac{}{x : A \vdash x : A}$$

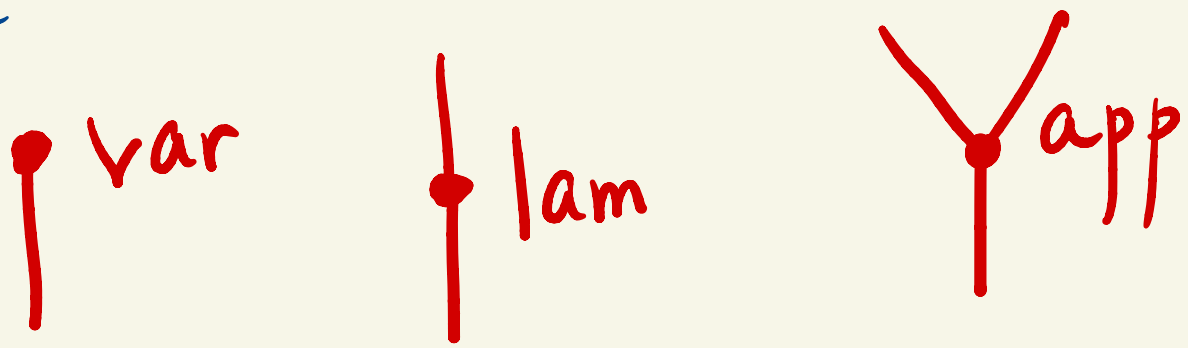
$$\text{lam}\ \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \Rightarrow B}$$

introduction of $\Rightarrow$

$$\text{app}\ \frac{\Gamma \vdash M : A \Rightarrow B \quad \Delta \vdash P : A}{\Gamma, \Delta \vdash \text{App}(M, P) : B}$$

elimination of $\Rightarrow$

have different variables in the context



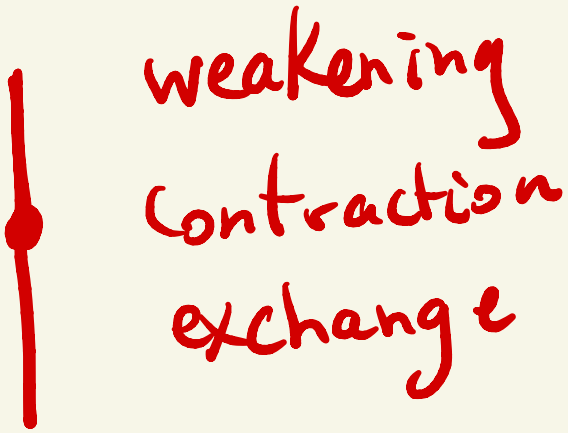
the three "logical" rules

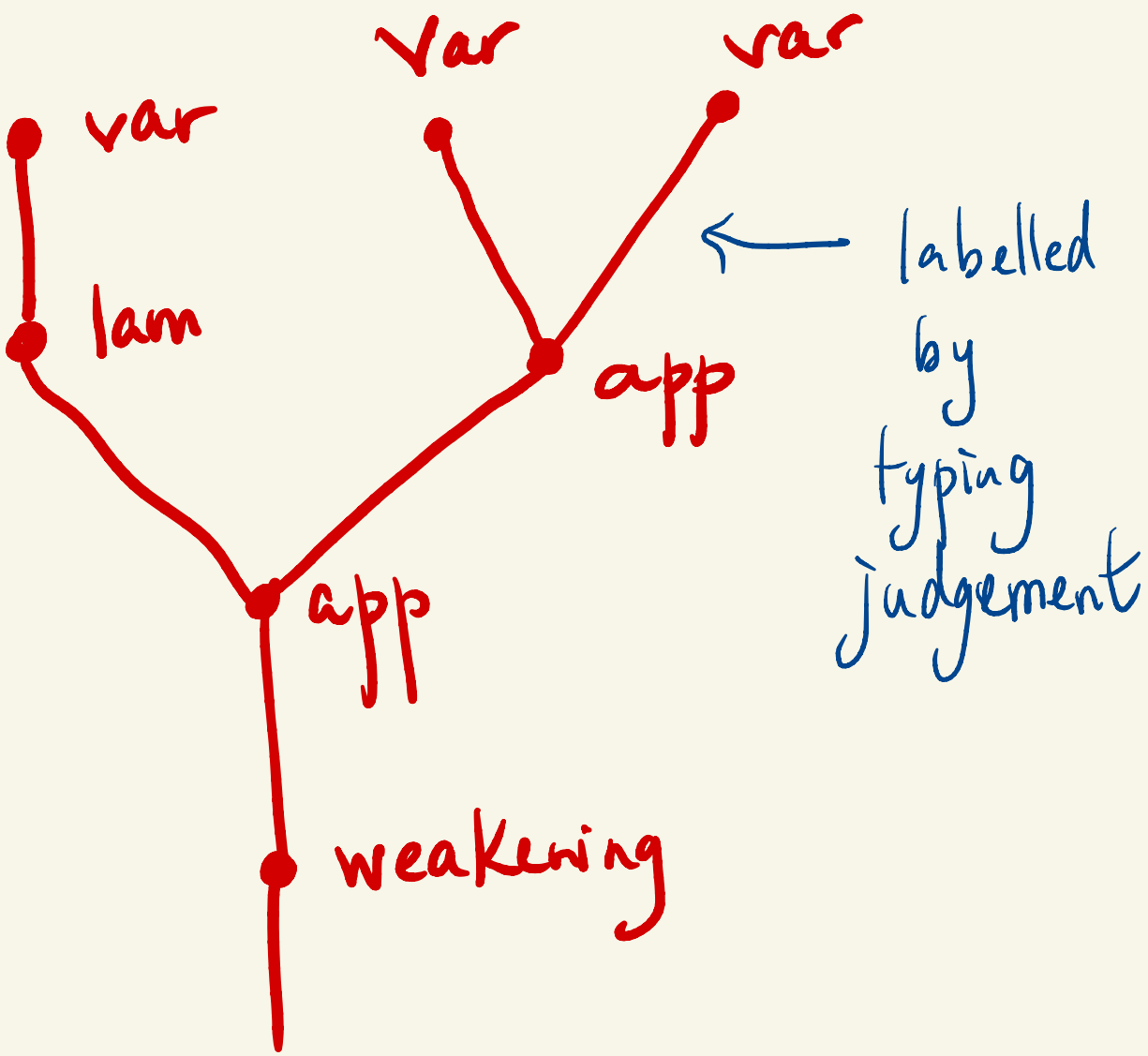
Structural rules

$$\frac{\Gamma \vdash M : B}{\Gamma, x : A \vdash M : B}\ \text{weakening}$$

$$\frac{\Gamma, x : A, y : A \vdash M : B}{\Gamma, z : A \vdash M[x, y := z] : B}\ \text{contraction}$$

$$\frac{\Gamma, x : A, y : B, \Delta \vdash M : C}{\Gamma, y : B, x : A, \Delta \vdash M : C}\ \text{exchange}$$





the shape of a derivation tree