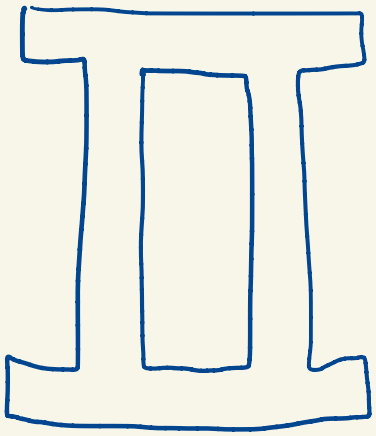
Lambda-Calculus & Categories 6

9 November 2020

λ-calculus

an introduction



β-rewriting in the λ-calculus

we studied the occurrences of a λ-term M

now we want to "rewrite" it using the intuition that a function $\lambda x.M$ is applied to an argument N

$$App(\lambda x.M, N) \xrightarrow[\beta\text{-rule}]{} M[x := N]$$

the β-rule is defined using a notion of <u>substitution</u> (capture-free) of a λ-term N for a free variable x in a λ-term M, what we write:

$$M[x := N]$$

Substitution is defined by induction on M:

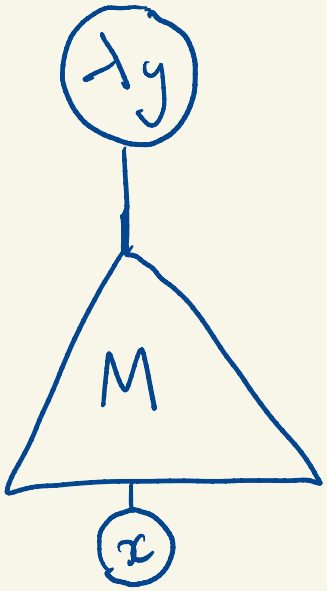
$$x[x := N] = N$$

$$y[x := N] = y \quad \text{when } y \neq x.$$

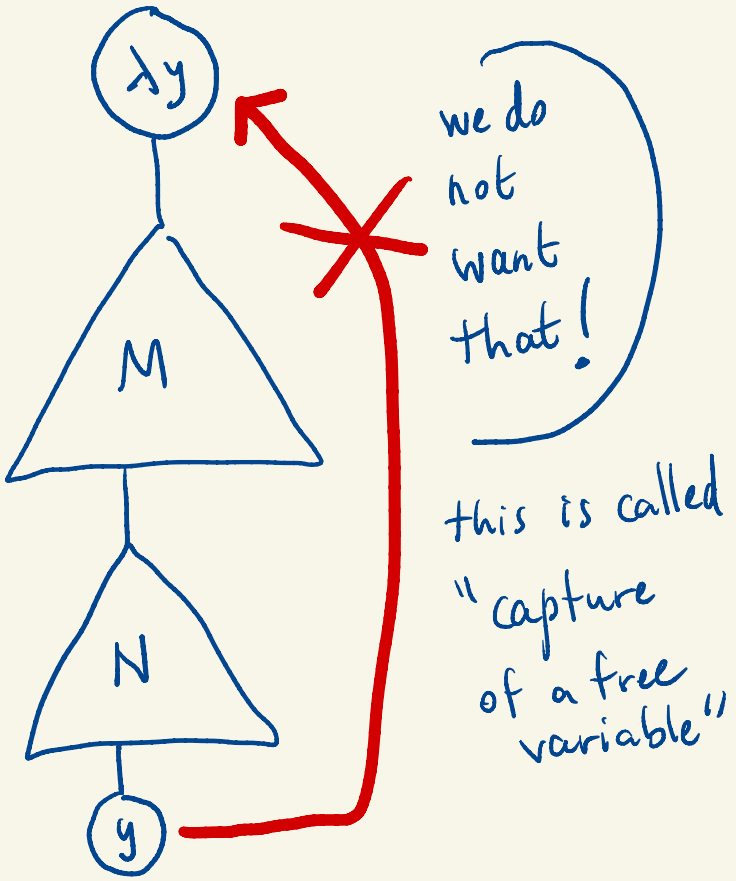
$$App(P, Q)[x := N] = App(P[x := N], Q[x := N])$$

$$(\lambda y . M)[x := N] = \lambda y . (M[x := N])$$

when $y \neq x$ and moreover y is not a free variable in N.



substitute
$\longmapsto$
x by N



this requirement
that the variable y is not free in N
is not problematic in practice
because we can always replace $\lambda y.M$
by $\lambda z.M'$ α-equivalent to $\lambda y.M$
where z is a "fresh" variable
not appearing in N. (or at least as a free variable of N)

this induces a <u>total</u> function

which transports every λ-term M

(up to α-conversion)

to a λ-term $M[x := N]$

(up to α-conversion)

<u>Illustration</u>:

$$\lambda y.x\ [x := App(f,z)]$$

$$= \lambda y.App(f,z) \quad (f \neq y \quad z \neq y)$$

$$\lambda y.x\ [x := App(f,y)]$$

$$= \lambda z.App(f,y)$$

here, the choice of the fresh variable z does not matter (as long as "it's fresh!") up to α-conversion

the β-rule is defined as:

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

example: I, K, Δ

$$I = \lambda x.x$$

$N = x[x := N]$

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

this justifies to call $I = \lambda x.x$ the identity combinator

$$K = \lambda x.\lambda y.x$$

the constant function

because y is not free in P

$$App(App(K, P), Q) \xrightarrow{\beta} App(\lambda y.P, Q)$$

sometimes written KPQ

no capture of a free variable by y

$$(\lambda x.\lambda y.x)PQ \xrightarrow{\beta} (\lambda y.P)Q$$

$$App(\lambda z.P, Q)$$

$$App(\lambda y.P, Q) \xrightarrow{\beta} P$$

because there is no free occurrence of the variable y in P.

$$K\,P\,Q \to (\lambda y.P)\,Q \to P$$

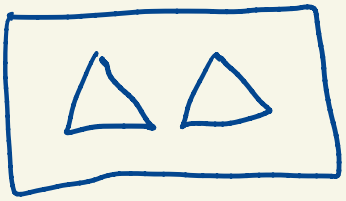
this justifies to call K "first projection" since K takes two arguments P and Q and returns the first one; while it "erases" the second argument.

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

the duplicator combinator

$$App(\Delta, P) \xrightarrow{\beta} App(P, P)$$

Δ: argument; P: function; $App(P, P)$: applied to itself!



argument

argument duplicated

$$App(\Delta, \Delta) \xrightarrow{\beta} App(\Delta, \Delta)$$

this means that $App(\Delta, \Delta)$ rewrites to itself!

this exhibits a "loop" in the λ-calculus

sometimes this combinator is noted

$$\Omega = App(\Delta, \Delta)$$

$$\Omega \xrightarrow{\beta} \Omega$$

$$\Omega \circlearrowright \beta$$

a "looping" combinator that "computes for ever"

The notion of β-redex

① restriction

every occurrence $o \in \mathrm{occ}(M)$ of a λ-term M induces a λ-term noted $M_{|o}$ and called "the restriction of M along the occurrence o"

$M_{|o}$ is defined by induction on o:

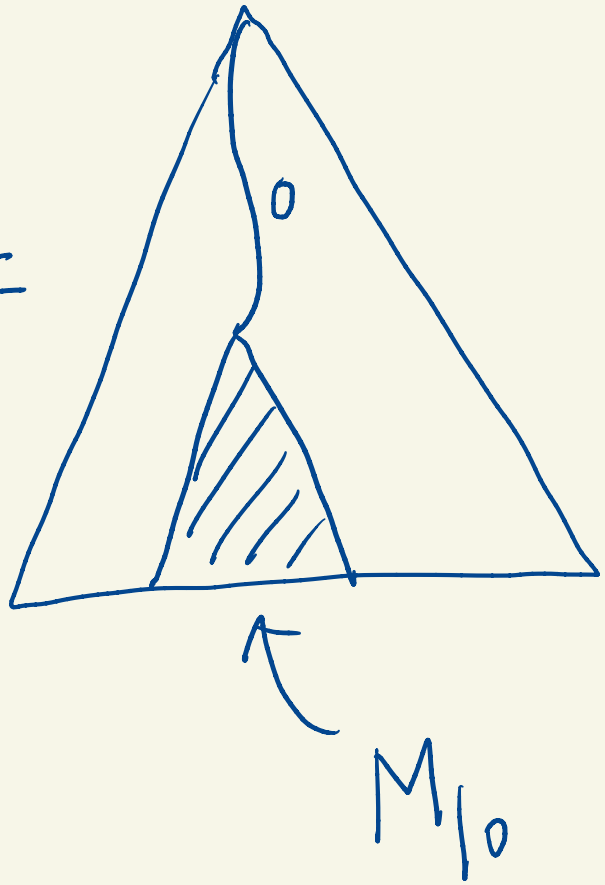
$$M_{|\varepsilon} = M$$

$$\mathrm{App}(M,N)_{|\mathrm{fun}\cdot o} = M_{|o}$$

$$\mathrm{App}(M,N)_{|\mathrm{arg}\cdot o} = N_{|o}$$

$$(\lambda x.M)_{|body \cdot 0} = M_{|0}$$

$M =$



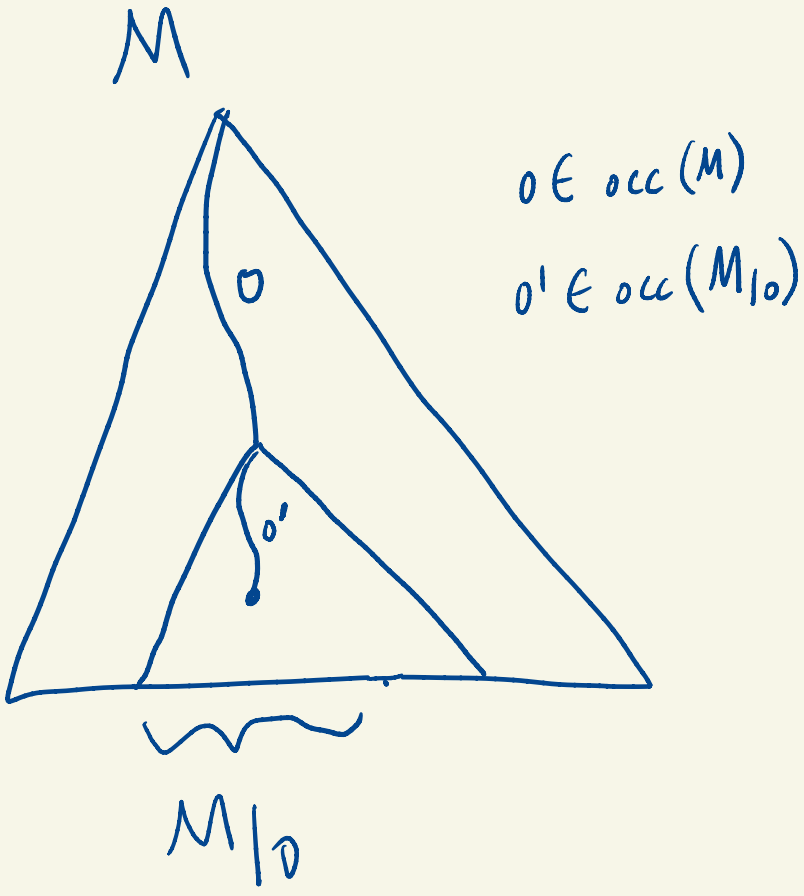
Remark: the notion of restriction depends on the choice of λ-term M in its α-equivalence class.

$$(\lambda x.x)_{|body} = x \qquad (\lambda y.y)_{|body} = y$$

Property every occurrence $o \in occ(M)$ induces a concatenation function

$$occ(M_{|o}) \longrightarrow occ(M)$$

$$o' \longmapsto o \cdot o'$$



$$\forall\, o' \in occ(M_{|o}),\quad o \cdot o' \in occ(M)$$

def. a β-redex is a pair

$$(M, o)$$

consisting of a λ-term M

(up to α-conversion)

and an occurrence $o \in \mathrm{occ}(M)$

of the λ-term M

such that the restriction

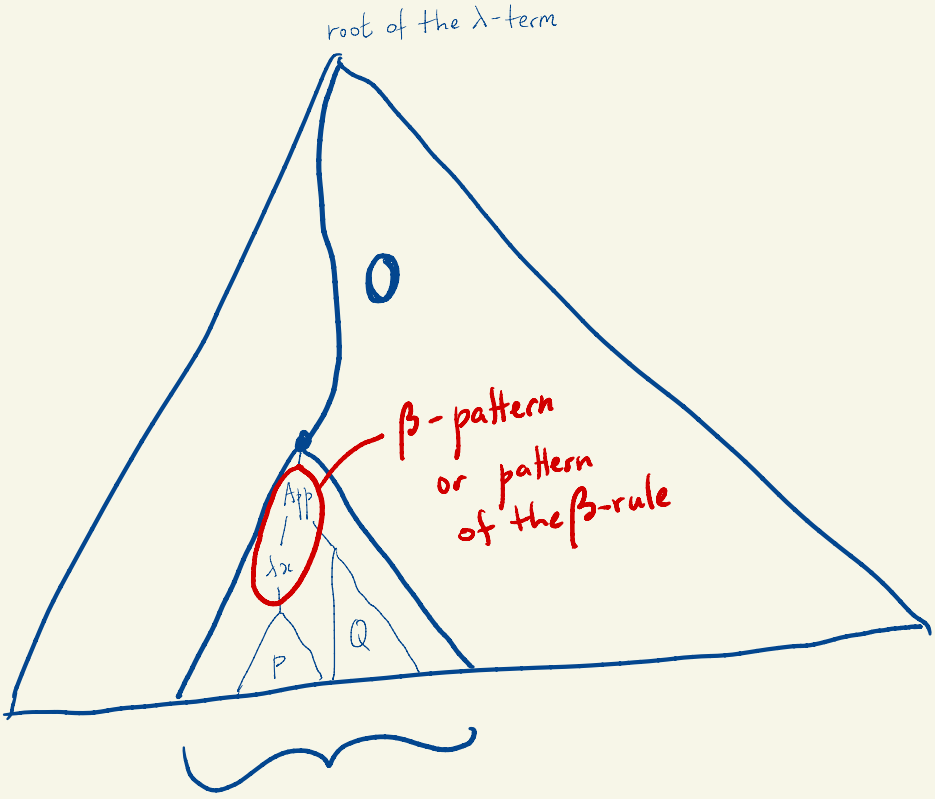
$$M|_o$$

is of the form a "β-pattern":

$$M|_o = \mathrm{App}(\lambda x.P, Q)$$

Illustration :

$M =$



$M_{|0}$

$App(\lambda x.P, Q)$

Every β-redex (M, o)
induces a β-rewriting step

$$M \longrightarrow N$$

where N is defined as:

$$C[P[x := Q]]$$

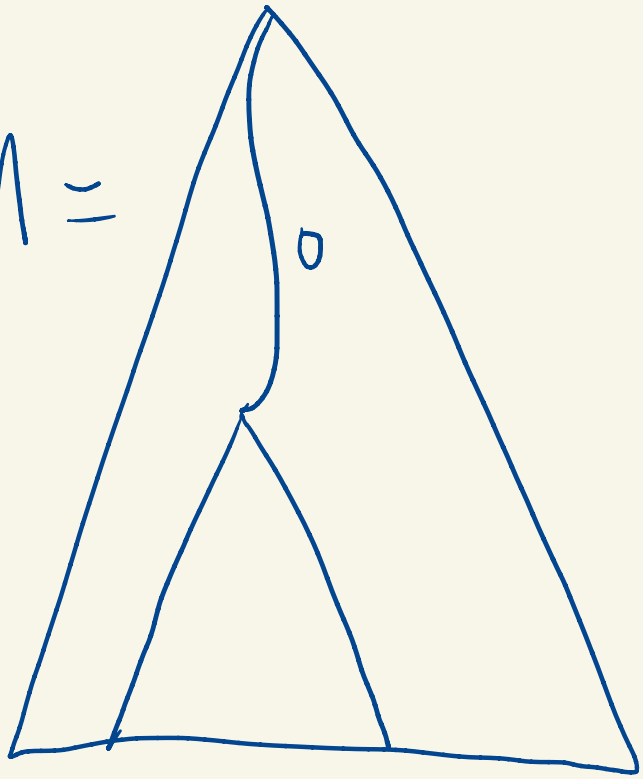
$P[x := Q]$: usual substitution (no capture of variable)

$C[\ldots]$: context substitution (possible capture of variable)

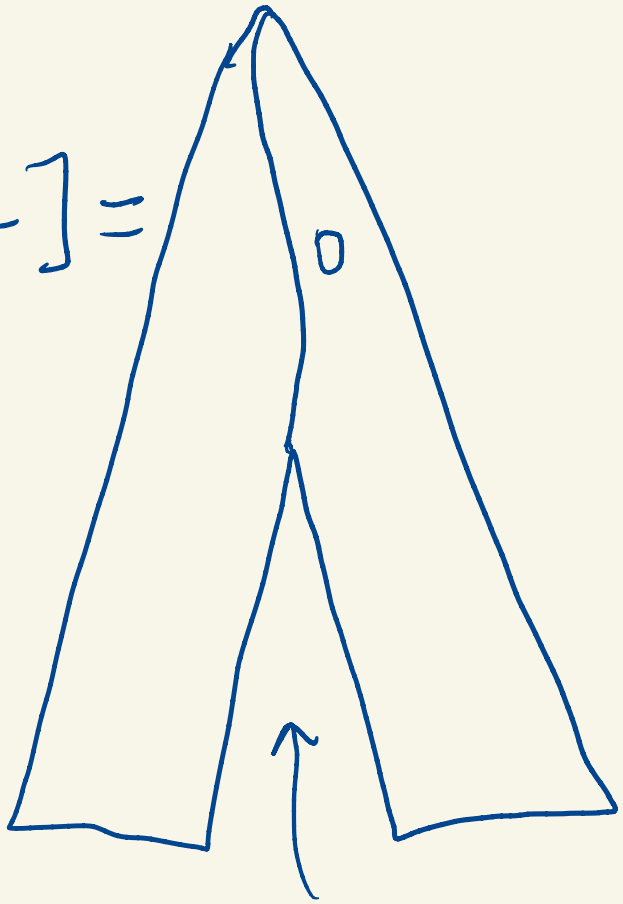
where $C[-]$ is the context such that

$$M = C[App(\lambda x.P, Q)]$$

$M =$



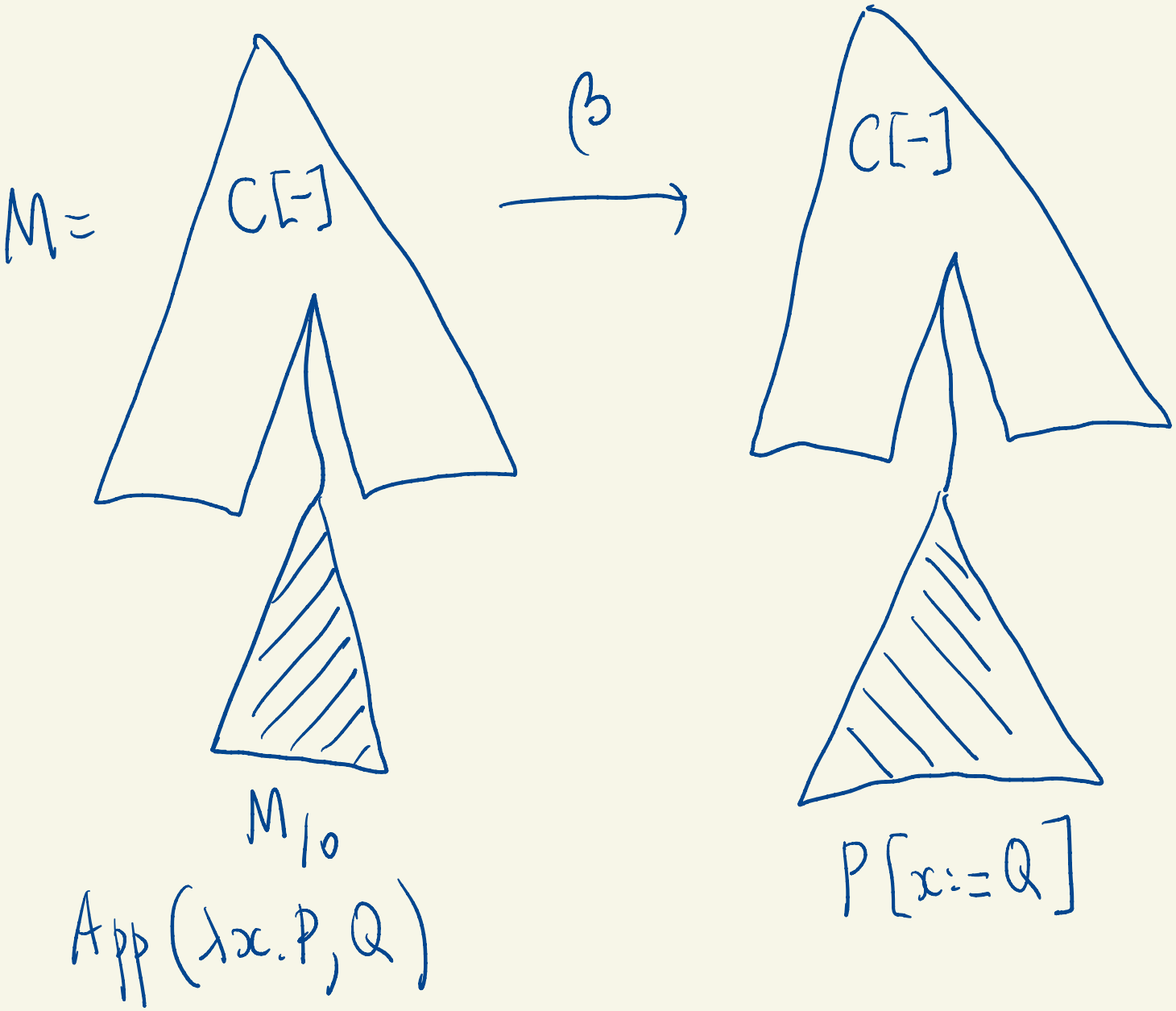
$C[-] =$

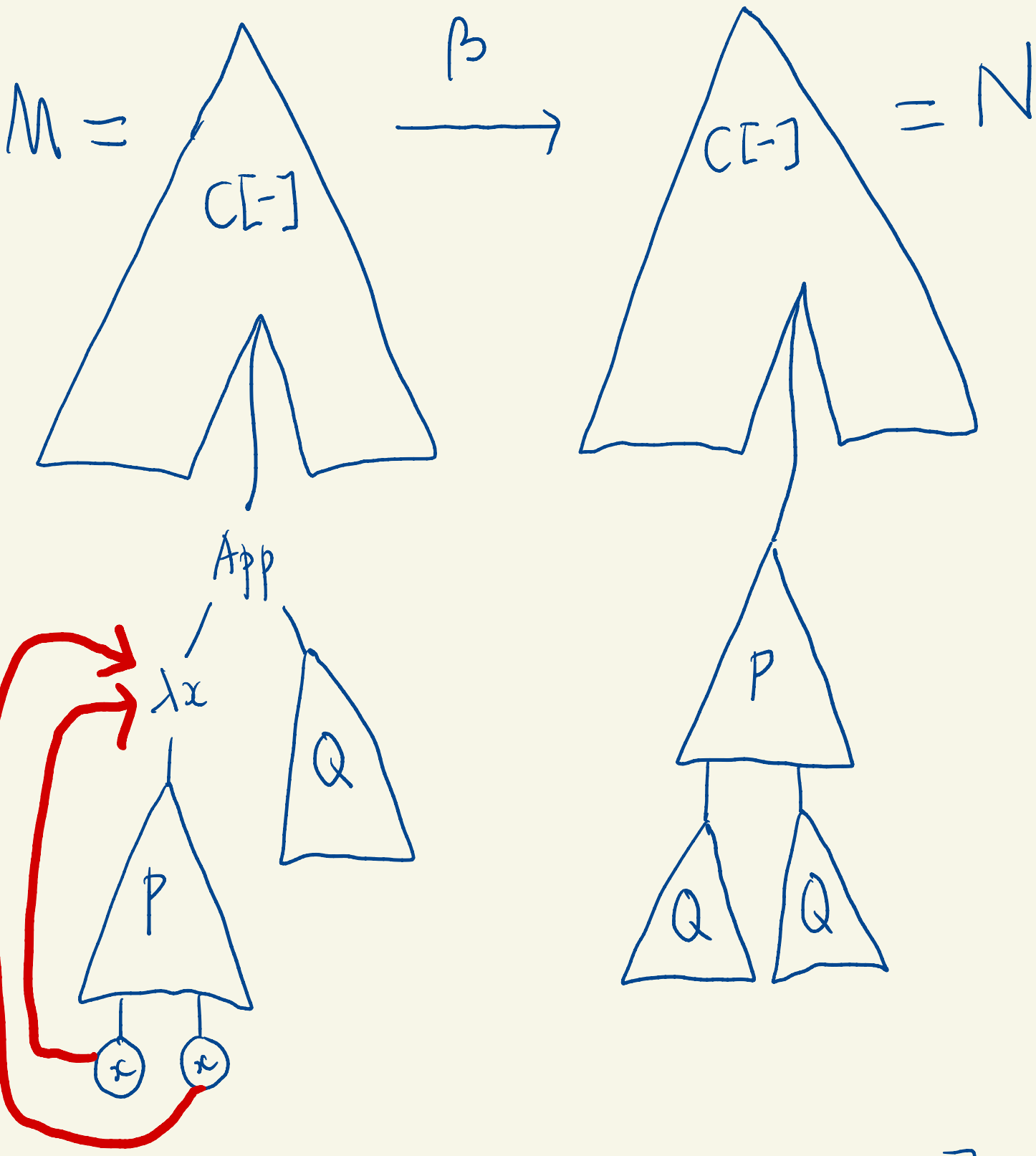


$$M_{|o} = App(\lambda x.P, Q)$$

$C[-] =$ λ-term with a "hole"

M is decomposed in $M = C[M_{10}]$





$App(\lambda x.P, Q)$

$P[x := Q]$

A nice example

(which justifies the notion of β-redex)

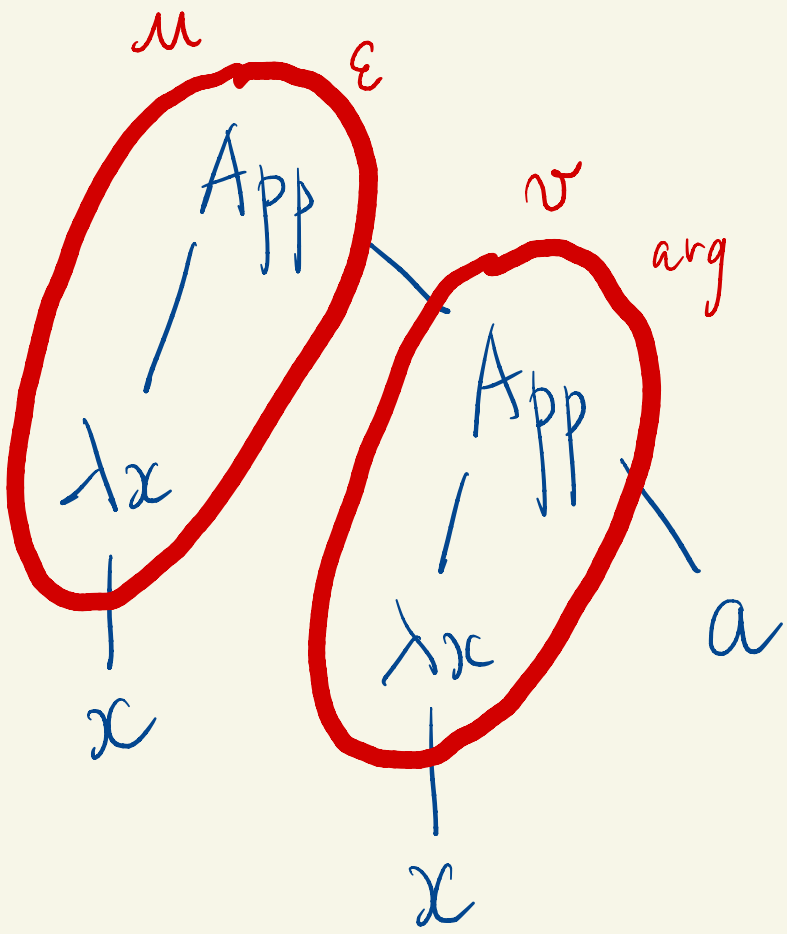
$$I = \lambda x.x$$

$$M = App(I, App(I, a))$$

how many β-redexes?

two of them

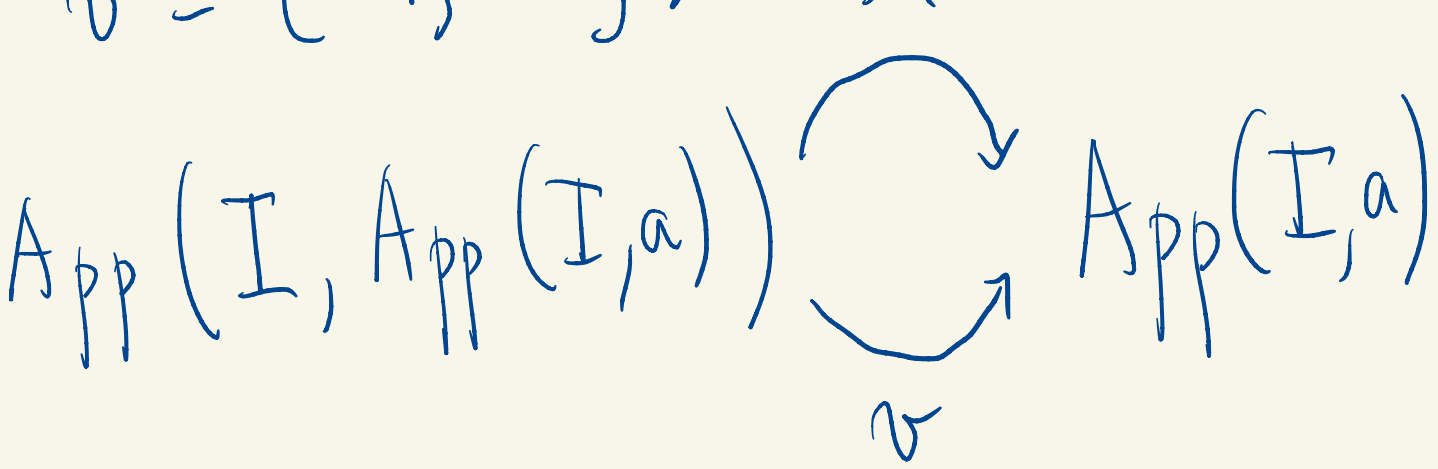
$M =$



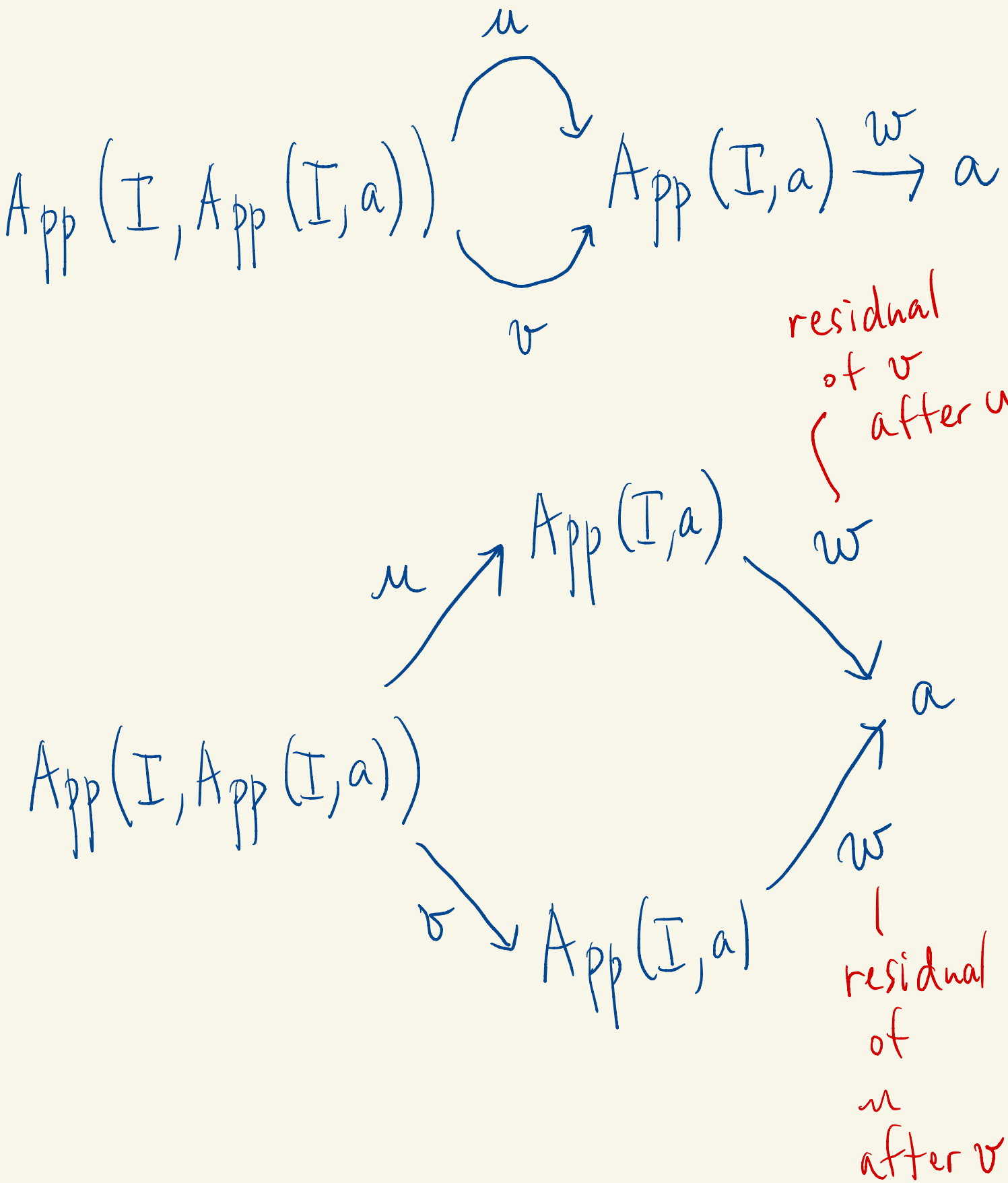
$u = (M, \varepsilon)$

$v = (M, arg)$

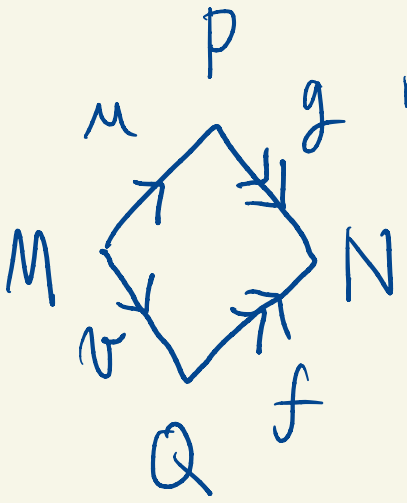
two β-redexes



β-redexes = "not just the what, also the where"


App(I, App(I, a))
u
v
App(I, a)
w
a
residual of v after u
u
App(I, a)
w
App(I, App(I, a))
a
v
App(I, a)
w
residual of u after v

Algebraic account of the Church-Rosser property



g rewrites the "residuals" of v after u

f rewrites the "residuals" of u after v

confluence property

$\forall f, g$

$\exists f', g'$