

# Lambda calculus

# Notions of computability

- ▶ Church (1936):  $\lambda$ -calculus
- ▶ Turing (1936): Turing machines.

Turing showed that the two very different approaches determine the same class of computable functions. Hence:

**Church-Turing Thesis.** Every algorithm [in intuitive sense of Lect. 1] can be realized as a Turing machine.

Notation for function definitions in mathematical discourse:

NAMED

" let  $f$  be the function  $f(x) = x^2 + x + 1 \dots [f] \dots$ "

ANONYMOUS

" the function  $x \mapsto x^2 + x + 1 \dots$ "

" the function  $\lambda x. x^2 + x + 1 \dots$ "

  
LAMBDA NOTATION

# $\lambda$ -Terms, $M$

are built up from a given, countable collection of

- ▶ variables  $x, y, z, \dots$

by two operations for forming  $\lambda$ -terms:

- ▶  $\lambda$ -abstraction:  $(\lambda x.M)$   
(where  $x$  is a variable and  $M$  is a  $\lambda$ -term)
- ▶ application:  $(M M')$   
(where  $M$  and  $M'$  are  $\lambda$ -terms).

Some random examples of  $\lambda$ -terms:

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

# $\lambda$ -Terms, $M$

## Notational conventions:

- ▶  $(\lambda x_1 x_2 \dots x_n. M)$  means  $(\lambda x_1. (\lambda x_2 \dots (\lambda x_n. M) \dots))$
- ▶  $(M_1 M_2 \dots M_n)$  means  $(\dots (M_1 M_2) \dots M_n)$   
(i.e. application is left-associative)
- ▶ drop outermost parentheses and those enclosing the body of a  $\lambda$ -abstraction. E.g. write  $(\lambda x. (x(\lambda y. (y x))))$  as  $\lambda x. x(\lambda y. y x)$ .
- ▶  $x \# M$  means that the variable  $x$  does not occur anywhere in the  $\lambda$ -term  $M$ .

# Free and bound variables

In  $\lambda x.M$ , we call  $x$  the **bound variable** and  $M$  the **body** of the  $\lambda$ -abstraction.

An occurrence of  $x$  in a  $\lambda$ -term  $M$  is called

- ▶ **binding** if in between  $\lambda$  and  $.$   
(e.g.  $(\lambda x.y x) x$ )
- ▶ **bound** if in the body of a binding occurrence of  $x$   
(e.g.  $(\lambda x.y x) x$ )
- ▶ **free** if neither binding nor bound  
(e.g.  $(\lambda x.y x)x$ ).

# Free and bound variables

Sets of **free** and **bound** variables:

$$\begin{aligned}FV(x) &= \{x\} \\FV(\lambda x.M) &= FV(M) - \{x\} \\FV(MN) &= FV(M) \cup FV(N) \\BV(x) &= \emptyset \\BV(\lambda x.M) &= BV(M) \cup \{x\} \\BV(MN) &= BV(M) \cup BV(N)\end{aligned}$$

E.g.  $FV((\lambda x.yx)x) = \{x, y\}$   
 $BV((\lambda x.yx)x) = \{x\}$

# Free and bound variables

Sets of **free** and **bound** variables:

$$\begin{aligned}FV(x) &= \{x\} \\FV(\lambda x.M) &= FV(M) - \{x\} \\FV(MN) &= FV(M) \cup FV(N) \\BV(x) &= \emptyset \\BV(\lambda x.M) &= BV(M) \cup \{x\} \\BV(MN) &= BV(M) \cup BV(N)\end{aligned}$$

If  $FV(M) = \emptyset$ ,  $M$  is called a **closed term**, or **combinator**.

$$\text{E.g. } FV(\lambda y. \lambda z. (\lambda x. yz)x) = \emptyset$$



## $\alpha$ -Equivalence $M =_{\alpha} M'$

$\lambda x.M$  is intended to represent the function  $f$  such that

$$f(x) = M \text{ for all } x.$$

So the name of the bound variable is immaterial: if  $M' = M\{x'/x\}$  is the result of taking  $M$  and changing all occurrences of  $x$  to some variable  $x' \# M$ , then  $\lambda x.M$  and  $\lambda x'.M'$  both represent the same function.

For example,  $\lambda x.x$  and  $\lambda y.y$  represent the same function (the identity function).

# $\alpha$ -Equivalence $M =_{\alpha} M'$

is the binary relation inductively generated by the rules:

$$\frac{}{x =_{\alpha} x} \qquad \frac{z \# (MN) \quad M\{z/x\} =_{\alpha} N\{z/y\}}{\lambda x.M =_{\alpha} \lambda y.N}$$

$$\frac{M =_{\alpha} M' \quad N =_{\alpha} N'}{MN =_{\alpha} M'N'}$$

where  $M\{z/x\}$  is  $M$  with all occurrences of  $x$  replaced by  $z$ .

# $\alpha$ -Equivalence $M =_{\alpha} M'$

For example:

$$\lambda \underline{x}. (\lambda \underline{x x'} . \underline{x}) x' =_{\alpha} \lambda \underline{y}. (\lambda x x' . x) x'$$

because

# $\alpha$ -Equivalence $M =_{\alpha} M'$

For example:

$$\lambda x. (\lambda x x'. x) x' =_{\alpha} \lambda y. (\lambda x x'. x) x'$$

because

$$(\lambda z x'. z) x' =_{\alpha} (\lambda x x'. x) x'$$

because

# $\alpha$ -Equivalence $M =_{\alpha} M'$

For example:

$$\lambda x. (\lambda x x'. x) x' =_{\alpha} \lambda y. (\lambda x x'. x) x'$$

because

$$(\lambda z x'. z) x' =_{\alpha} (\lambda x x'. x) x'$$

because

$$\lambda \underline{z} x'. z =_{\alpha} \lambda \underline{x} x'. x \text{ and } x' =_{\alpha} x'$$

because

# $\alpha$ -Equivalence $M =_{\alpha} M'$

For example:

$$\lambda x. (\lambda x x'. x) x' =_{\alpha} \lambda y. (\lambda x x'. x) x'$$

because

$$(\lambda z x'. z) x' =_{\alpha} (\lambda x x'. x) x'$$

because

$$\lambda z x'. z =_{\alpha} \lambda x x'. x \text{ and } x' =_{\alpha} x'$$

because

$$\lambda \underline{x'} . u =_{\alpha} \lambda \underline{x'} . u \text{ and } x' =_{\alpha} x'$$

because

# $\alpha$ -Equivalence $M =_{\alpha} M'$

For example:

$$\lambda x. (\lambda x x'. x) x' =_{\alpha} \lambda y. (\lambda x x'. x) x'$$

because

$$(\lambda z x'. z) x' =_{\alpha} (\lambda x x'. x) x'$$

because

$$\lambda z x'. z =_{\alpha} \lambda x x'. x \text{ and } x' =_{\alpha} x'$$

because

$$\lambda x'. u =_{\alpha} \lambda x'. u \text{ and } x' =_{\alpha} x'$$

because

$$u =_{\alpha} u \text{ and } x' =_{\alpha} x'.$$

# $\alpha$ -Equivalence $M =_{\alpha} M'$

**Fact:**  $=_{\alpha}$  is an equivalence relation (reflexive, symmetric and transitive).

We do not care about the particular names of bound variables, just about the distinctions between them. So  $\alpha$ -equivalence classes of  $\lambda$ -terms are more important than  $\lambda$ -terms themselves.

- ▶ Textbooks (**and these lectures**) suppress any notation for  $\alpha$ -equivalence classes and refer to an equivalence class via a representative  $\lambda$ -term (look for phrases like “we identify terms up to  $\alpha$ -equivalence” or “we work up to  $\alpha$ -equivalence”).
- ▶ For implementations and computer-assisted reasoning, there are various devices for picking canonical representatives of  $\alpha$ -equivalence classes (e.g. de Bruijn indexes, graphical representations, ...).



# $\beta$ -Reduction

Recall that  $\lambda x.M$  is intended to represent the function  $f$  such that  $f(x) = M$  for all  $x$ . We can regard  $\lambda x.M$  as a function on  $\lambda$ -terms via substitution: map each  $N$  to  $M[N/x]$ .

So the natural notion of computation for  $\lambda$ -terms is given by stepping from a

$\beta$ -redex  $(\lambda x.M)N$

to the corresponding

$\beta$ -reduct  $M[N/x]$

# Substitution $N[M/x]$

$$\begin{aligned}x[M/x] &= M \\y[M/x] &= y \quad \text{if } y \neq x \\(\lambda y.N)[M/x] &= \lambda y.N[M/x] \quad \text{if } y \# (M x) \\(N_1 N_2)[M/x] &= N_1[M/x] N_2[M/x]\end{aligned}$$

$N[M/x]$  = result of replacing all free occurrences of  $x$  in  $N$  with  $M$ , avoiding "capture" of free variables in  $M$  by  $\lambda$ -binders in  $N$

# Substitution $N[M/x]$

$$\begin{aligned}x[M/x] &= M \\y[M/x] &= y \quad \text{if } y \neq x \\(\lambda y.N)[M/x] &= \lambda y.N[M/x] \quad \text{if } y \# (M x) \\(N_1 N_2)[M/x] &= N_1[M/x] N_2[M/x]\end{aligned}$$

Side-condition  $y \# (M x)$  ( $y$  does not occur in  $M$  and  $y \neq x$ ) makes substitution “capture-avoiding”.

E.g. if  $x \neq y$

$$(\lambda y.x)[y/x] \neq \lambda y.y$$

# Substitution $N[M/x]$

$$\begin{aligned}x[M/x] &= M \\y[M/x] &= y \quad \text{if } y \neq x \\(\lambda y.N)[M/x] &= \lambda y.N[M/x] \quad \text{if } y \# (M x) \\(N_1 N_2)[M/x] &= N_1[M/x] N_2[M/x]\end{aligned}$$

Can always satisfy this  up to  $\alpha$ -equivalence

E.g. if  $x \neq y \neq z \neq x$

$$(\lambda y.x)[y/x] =_{\alpha} (\lambda z.x)[y/x] = \lambda z.y$$

In fact  $N \mapsto N[M/x]$  induces a totally defined function from the set of  $\alpha$ -equivalence classes of  $\lambda$ -terms to itself.

||

$$\lambda x. (\lambda z. z) y x \left[ \lambda z. y / y \right]$$

=

$$\lambda x. (\lambda z. z) y x \quad [ \quad \lambda z. y / y \quad ]$$

no possible  
capture

$$\lambda x. (\lambda z. z) y x \left[ \lambda z. y / y \right]$$
$$= \lambda x. (\lambda z. z) (\lambda z. y) x$$

---

$$\lambda x. (\lambda u. u) x y \left[ \lambda y. x / y \right]$$
$$=$$

$$\lambda x. (\lambda z. z) y x \quad [ \lambda x. y / y ]$$
$$= \lambda x. (\lambda z. z) (\lambda x. y) x$$

---

$$\lambda x. (\lambda u. u) x y \quad [ \lambda y. x / y ] \quad \text{possible capture}$$
$$=$$



$$\lambda x. (\lambda z. z) y x \ [ \ \lambda x. y / y \ ]$$

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


---

$$\lambda x. (\lambda u. u) x y \ [ \ \lambda y. x / y \ ] \quad \text{possible capture...}$$

$$=_{\alpha} \lambda z. (\lambda u. u) z y \ [ \ \lambda y. x / y \ ] \quad \text{...}\alpha\text{-convert to avoid}$$

$$\lambda x. (\lambda z. z) y x \left[ \lambda x. y / y \right]$$

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


---

$$\lambda x. (\lambda u. u) x y \left[ \lambda y. x / y \right]$$

possible capture...

$$\stackrel{\alpha}{=} \lambda z. (\lambda u. u) z y \left[ \lambda y. x / y \right]$$

... $\alpha$ -convert to avoid

$$= \lambda z. (\lambda u. u) z (\lambda y. x)$$

# $\beta$ -Reduction

One-step  $\beta$ -reduction,  $M \rightarrow M'$ :

$$\frac{}{(\lambda x.M)N \rightarrow M[N/x]} \qquad \frac{M \rightarrow M'}{\lambda x.M \rightarrow \lambda x.M'}$$

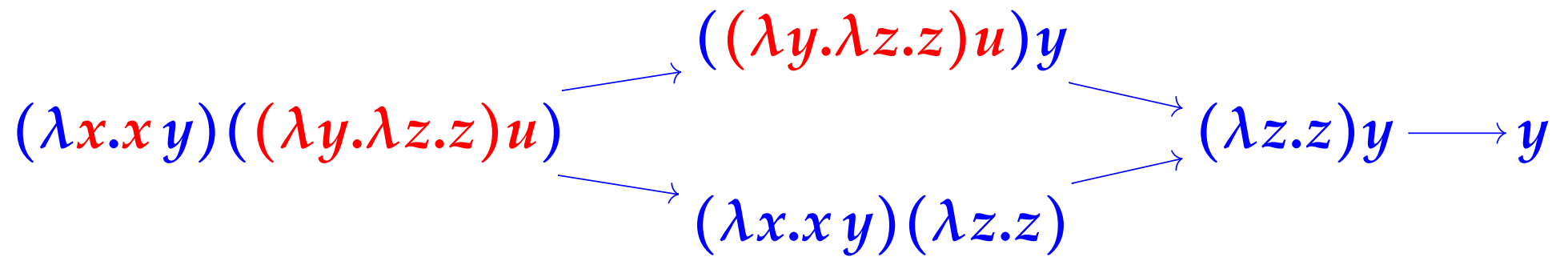
$$\frac{M \rightarrow M'}{MN \rightarrow M'N}$$

$$\frac{M \rightarrow M'}{NM \rightarrow NM'}$$

$$\frac{N =_{\alpha} M \quad M \rightarrow M' \quad M' =_{\alpha} N'}{N \rightarrow N'}$$

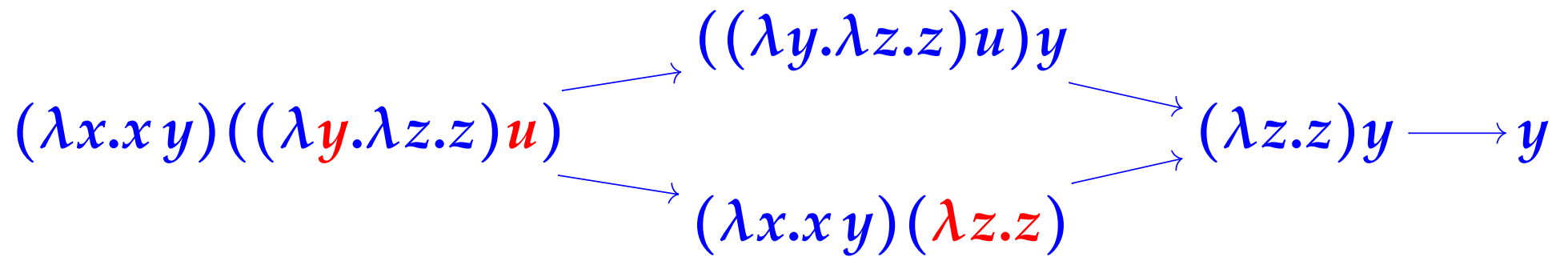
# $\beta$ -Reduction

E.g.



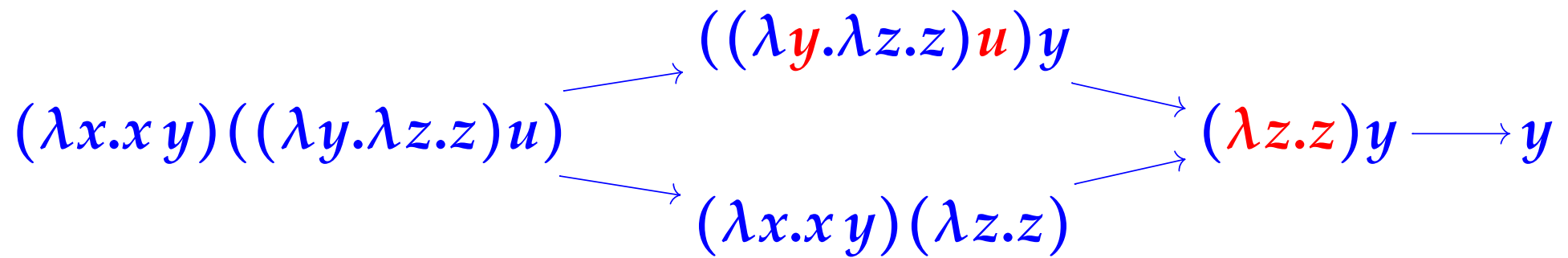
# $\beta$ -Reduction

E.g.



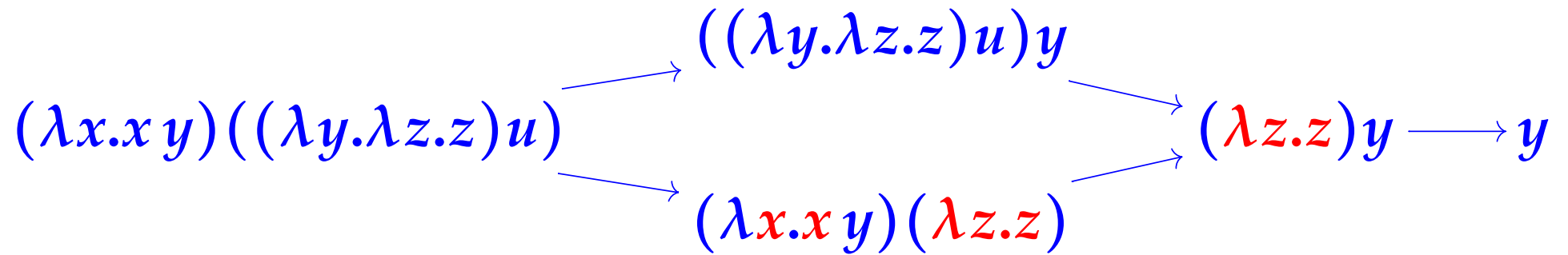
# $\beta$ -Reduction

E.g.



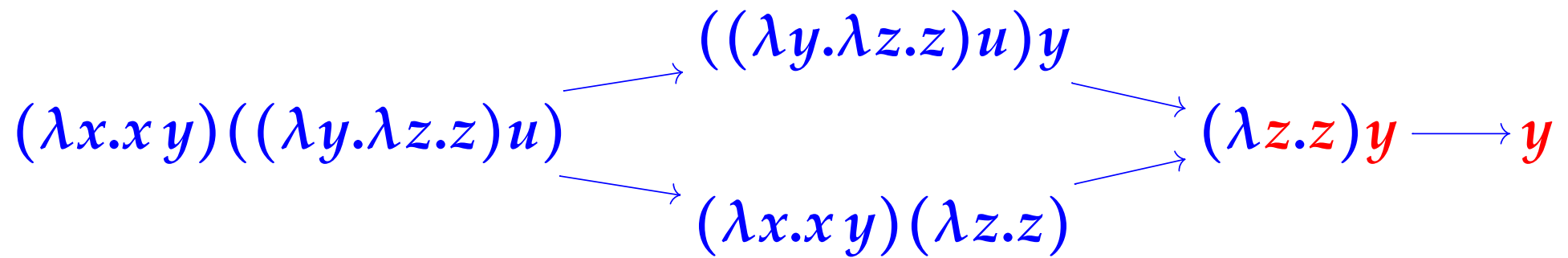
# $\beta$ -Reduction

E.g.



# $\beta$ -Reduction

E.g.





Many-step  $\beta$ -reduction,  $M \rightarrow\!\!\rightarrow M'$ :

$$\frac{M =_{\alpha} M'}{M \rightarrow\!\!\rightarrow M'} \quad \left[ \frac{M \rightarrow M'}{M \rightarrow\!\!\rightarrow M'} \right] \quad \frac{M \rightarrow\!\!\rightarrow M' \quad M' \rightarrow M''}{M \rightarrow\!\!\rightarrow M''}$$

(no steps)                      (1 step)                      (1 more step)

E.g.

$$(\lambda x. x y) ((\lambda y z. z) u) \rightarrow\!\!\rightarrow y$$

# $\beta$ -Conversion $M =_{\beta} N$

Informally:  $M =_{\beta} N$  holds if  $N$  can be obtained from  $M$  by performing zero or more steps of  $\alpha$ -equivalence,  $\beta$ -reduction, or  $\beta$ -expansion (= inverse of a reduction).

E.g.  $u((\lambda x y. v x)y) =_{\beta} (\lambda x. u x)(\lambda x. v y)$

because  $(\lambda x. u x)(\lambda x. v y) \rightarrow u(\lambda x. v y)$

and so we have

$u((\lambda x y. v x)y) =_{\alpha} u((\lambda x y'. v x)y)$

# $\beta$ -Conversion $M =_{\beta} N$

Informally:  $M =_{\beta} N$  holds if  $N$  can be obtained from  $M$  by performing zero or more steps of  $\alpha$ -equivalence,  $\beta$ -reduction, or  $\beta$ -expansion (= inverse of a reduction).

E.g.  $u((\lambda x y. v x)y) =_{\beta} (\lambda x. u x)(\lambda x. v y)$

because  $(\lambda x. u x)(\lambda x. v y) \rightarrow u(\lambda x. v y)$

and so we have

$$\begin{aligned} u((\lambda x y. v x)y) &=_{\alpha} u((\lambda x y'. v x)y) \\ &\rightarrow u(\lambda y'. v y) && \text{reduction} \end{aligned}$$

# $\beta$ -Conversion $M =_{\beta} N$

Informally:  $M =_{\beta} N$  holds if  $N$  can be obtained from  $M$  by performing zero or more steps of  $\alpha$ -equivalence,  $\beta$ -reduction, or  $\beta$ -expansion (= inverse of a reduction).

E.g.  $u((\lambda x y. v x)y) =_{\beta} (\lambda x. u x)(\lambda x. v y)$

because  $(\lambda x. u x)(\lambda x. v y) \rightarrow u(\lambda x. v y)$

and so we have

$$\begin{aligned} u((\lambda x y. v x)y) &=_{\alpha} u((\lambda x y'. v x)y) \\ &\rightarrow u(\lambda y'. v y) && \text{reduction} \\ &=_{\alpha} u(\lambda x. v y) \end{aligned}$$

# $\beta$ -Conversion $M =_{\beta} N$

Informally:  $M =_{\beta} N$  holds if  $N$  can be obtained from  $M$  by performing zero or more steps of  $\alpha$ -equivalence,  $\beta$ -reduction, or  $\beta$ -expansion (= inverse of a reduction).

E.g.  $u((\lambda x y. v x)y) =_{\beta} (\lambda x. u x)(\lambda x. v y)$

because  $(\lambda x. u x)(\lambda x. v y) \rightarrow u(\lambda x. v y)$

and so we have

$$\begin{aligned} u((\lambda x y. v x)y) &=_{\alpha} u((\lambda x y'. v x)y) \\ &\rightarrow u(\lambda y'. v y) && \text{reduction} \\ &=_{\alpha} u(\lambda x. v y) \\ &\leftarrow (\lambda x. u x)(\lambda x. v y) && \text{expansion} \end{aligned}$$

# $\beta$ -Conversion $M =_{\beta} N$

is the binary relation inductively generated by the rules:

$$\frac{M =_{\alpha} M'}{M =_{\beta} M'}$$

$$\frac{M \rightarrow M'}{M =_{\beta} M'}$$

$$\frac{M =_{\beta} M'}{M' =_{\beta} M}$$

$$\frac{M =_{\beta} M' \quad M' =_{\beta} M''}{M =_{\beta} M''}$$

$$\frac{M =_{\beta} M'}{\lambda x.M =_{\beta} \lambda x.M'}$$

$$\frac{M =_{\beta} M' \quad N =_{\beta} N'}{MN =_{\beta} M'N'}$$