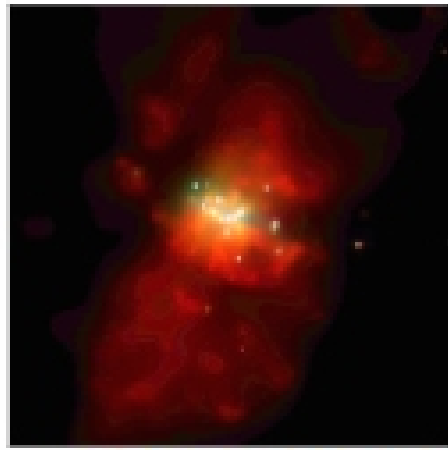


Class 28: The Meaning of Truth



λ -calculus

Alonzo Church, 1940

(LISP was developed from λ -calculus,
not the other way round.)

$term = variable$

| $term term$

| $(term)$

| $\lambda variable . term$

Reduction (Uninteresting Rules)

$\lambda y. M \rightarrow \lambda v. (M [y \mapsto v])$

where v does not occur in M .

$M \rightarrow M$

$M \rightarrow N \Rightarrow PM \rightarrow PN$

$M \rightarrow N \Rightarrow MP \rightarrow NP$

$M \rightarrow N \Rightarrow \lambda x. M \rightarrow \lambda x. N$

$M \rightarrow N$ and $N \rightarrow P \Rightarrow M \rightarrow P$

β -Reduction (the source of all computation)

$(\lambda x. M)N \rightarrow M [x \mapsto N]$

Evaluating Lambda Expressions

- *redex*: Term of the form $(\lambda x. M)N$
Something that can be β -reduced
- An expression is in *normal form* if it contains no redexes (*redices*).
- To evaluate a lambda expression, keep doing reductions until you get to normal form.

Recall Apply in Scheme

"To **apply** a procedure to a list of arguments, **evaluate** the procedure in a new environment that binds the formal parameters of the procedure to the arguments it is applied to."

- We've replaced environments with substitution.
- We've replaced **eval** with reduction.

Some Simple Functions

$I = \lambda x.x$

$C = \lambda xy.yx$

Abbreviation for $\lambda x.(\lambda y.yx)$

$CI = (\lambda x.(\lambda y.yx)) (\lambda x.x)$

$\rightarrow_{\beta} (\lambda y.y (\lambda x.x)) (\lambda x.x)$

$\rightarrow_{\beta} \lambda x.x (\lambda x.x)$

$\rightarrow_{\beta} \lambda x.x$

$= I$

Example

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

Try this one at home...

Alyssa P. Hacker's Answer

$(\lambda f. ((\lambda x.f(xx)) (\lambda x.f(xx)))) (\lambda z.z)$
 $\rightarrow_{\beta} (\lambda x.(\lambda z.z)(xx)) (\lambda x.(\lambda z.z)(xx))$
 $\rightarrow_{\beta} (\lambda z.z) (\lambda x.(\lambda z.z)(xx)) (\lambda x.(\lambda z.z)(xx))$
 $\rightarrow_{\beta} (\lambda x.(\lambda z.z)(xx)) (\lambda x.(\lambda z.z)(xx))$
 $\rightarrow_{\beta} (\lambda z.z) (\lambda x.(\lambda z.z)(xx)) (\lambda x.(\lambda z.z)(xx))$
 $\rightarrow_{\beta} (\lambda x.(\lambda z.z)(xx)) (\lambda x.(\lambda z.z)(xx))$
 $\rightarrow_{\beta} \dots$

Ben Bitdiddle's Answer

$(\lambda f. ((\lambda x.f(xx)) (\lambda x.f(xx)))) (\lambda z.z)$
 $\rightarrow_{\beta} (\lambda x.(\lambda z.z)(xx)) (\lambda x.(\lambda z.z)(xx))$
 $\rightarrow_{\beta} (\lambda x.xx) (\lambda x.(\lambda z.z)(xx))$
 $\rightarrow_{\beta} (\lambda x.xx) (\lambda x.xx)$
 $\rightarrow_{\beta} (\lambda x.xx) (\lambda x.xx)$
 $\rightarrow_{\beta} \dots$

Be Very Afraid!

- Some λ -calculus terms can be β -reduced forever!
- The order in which you choose to do the reductions might change the result!

Take on Faith (until CS655)

- All ways of choosing reductions that reduce a lambda expression to normal form will produce the same normal form (but some might never produce a normal form).
- If we always *apply the outermost lambda first*, we will find the normal form if there is one.
 - This is *normal order reduction* – corresponds to normal order (lazy) evaluation

