Lambda Calculus

Alonzo Church 1936, same year as Turing machines

Syntax of a Lambda Expression

i ::= a | b | c | d | e | ..... | a_0 | a_1 | a_2 | a_3 | a_4 | ..... *etc*

Λ	**::=**	i	identifier
	|	**λ** i **.** Λ	abstraction
	|	**(** Λ Λ **)**	application

In the last rule any kind of bracket may be used, it makes no difference.
Capital letters are usually used to represent any properly written lambda expression.

A few examples

x
(x y)
λx.y
λa.(b a)
λc.λd.(c λe.(d e))
(λx.(x x) λy.(y y))

λx.Y	may be thought of informally as a function. Its parameter is named x and the value returned is Y.
(F X)	may be thought of informally as applying the function F to the value X.
λx.λx.Z	may be thought of informally as a function with two parameters x and y where the value returned is Z.
((F X) Y)	may be thought of informally as a function of two parameters being called with X and Y as the values of those parameters.

Informal semantics with examples

α alpha reductions

Anywhere that something of the form λx.Y appears, the name x may be uniformly replaced throughout it with any other name that does not already appear in Y.

λa.(b a)	≡	λx.(b x)	
λx.x	≡	λa.a	
λx.y	≡	λa.y	
(a λb.(a b))	≡	(a λp.(a p))	
λa.(λa.(a b) λb.(a b))	≡	λx.(λa.(a b) λb.(x b))	[*]
λx.(x λb.(x b))	≡	λx.(x λy.(x y))	

β beta reductions

Anywhere that something of the form (λx.Y Z) appears, so long as there is no possible confusion between the names appearing in Y and Z, it may be replaced by a copy of Y in which every occurrence of x has been replaced by a copy of Z.

(λa.(b a) m)	≡	(b m)
(λa.(a b) m)	≡	(m b)
(λa.(a b) (a m))	≡	((a m) b)
(λa.a λx.x)	≡	λx.x
(λa.x λx.x)	≡	x
(λa.(a a) (x y))	≡	((x y) (x y))
λx.(λy.(z y) k)	≡	λx.(z k)

Multiple reductions

	(λa.(a (a b)) λx.(x x))
≡	(λx.(x x) (λx.(x x) b))
≡	([λx.(x x) b] [λx.(x x) b])
≡	((b b) [λx.(x x) b])
≡	((b b) (b b))

Notice that at the second and third steps there were two possible beta reductions. I could have chosen either and be guaranteed not to get different results.

	(λa.(a (a b)) λx.(x x))
≡	(λx.(x x) (λx.(x x) b))
≡	(λx.(x x) (b b))
≡	((b b) (b b))

or

	(λa.(a (a b)) λx.(x x))
≡	(λx.(x x) (λx.(x x) b))
≡	([λx.(x x) b] [λx.(x x) b])
≡	([λx.(x x) b] (b b))
≡	((b b) (b b))

[*] made it clear that the informal rules given above are not quite right. These are the exact rules for when α and β reductions are permitted and how they are performed.

First we need to define two terms, bound and free variable.

Does an identifier appear “bound” or “free” in a lambda expression, case by case:

x does not appear bound in x (itself).
x does not appear bound in y (some other identifier).
x appears bound in λx.A only if x appears free in A.
x appears bound in λy.A only if x appears bound in A.
x appears bound in (A B) if x appears bound in A or x appears bound in B.

x does appears free in x (itself).
x does not appear free in y (some other identifier).
x does not appear free in λx.A.
x appears free in λy.A only if x and y are different and x appears free in A.
x appears free in (A B) if x appears free in A or x appears free in B.

How substitutions are performed, {...}[.../...] is a three operand operator {A}[b/C] is the same as A except that every eligible appearance of b inside it is replaced by C. The rules are almost but not quite as follows:

{ x }[x / Z]	is	Z	
{ y }[x / Z]	is	y	(unless x and y are the same identifier)
{ λx.A }[x / Z]	is	λx.A	(unchanged)
{ λy.A }[x / Z]	is	λy.{ A }[x / Z]	(unless x and y are the same identifier)
{ (A B) }[x / Z]	is	({ A }[x / Z] { B }[x / Z])	

The reductions (or conversions)

λx.A	α≡	λy.{ A }[x / y]	only if y does not appear free in A.
(λx.A B)	β≡	{ A }[x / B]	only if x does not appear bound in A.

α reductions are only used to allow a β reduction that is forbidden by the second rule, or to show that two lambda expressions are the same. Two lambda expressions are the same if they can be α reduced to each other.

Those rules are good enough for just about everything, but not absolutely everything. A very simple example illustrates the problem:

λb.(a b)

and

λc.(a c)

are α-reducible to each other, so in every meaningful way they are the same thing. Anything that involves λb.(a b) should behave exactly the same if the λb.(a b) is replaced with λc.(a c).

(λa.λb.(a b) b) β≡ λb.(b b)

but

(λa.λc.(a c) b) β≡ λc.(b c)

and λb.(b b) and λc.(b c) can not be reduced to each other, except by ignoring the α-reduction rules.

The exact restriction on a β-reduction is

$(\lambda x.A\ B) \beta\equiv \{ A \}[x / B]$ only if

nothing that appears free in B
appears bound in A.

If a programmed system for performing reductions is being considered rather than doing it all by hand, collecting together all the variables that appear free in B and checking them all to make sure none appear bound in A can be annoying and inefficient. But that is no problem. The rules we have just seen are not quite perfect, and perfecting them simplifies the task.

The complete rules are best expressed by saying that

$$(\lambda x.A\ B) \beta\equiv \{ A \}[x / B]$$

always, with absolutely no restrictions, but then refining the definition of {...}[.../...] as follows:

$\{ x \}[x / Z]$	is Z	
$\{ y \}[x / Z]$	is y	(unless x and y are the same identifier)
$\{ (A\ B) \}[x / Z]$	is $(\ \{ A \}[x / Z]\ \{ B \}[x / Z]\ )$	
$\{ \lambda x.A \}[x / Z]$	is $\lambda x.A$	(unchanged)
$\{ \lambda y.A \}[x / Z]$	(when x and y are different identifiers)	
	is $\lambda y.\{ A \}[x / Z]$	if x does not appear free in A or y does not appear free in Z
	or $\lambda n.\{ A \}[y / n][x / Z]$	otherwise, where n is a totally new variable.

The first condition, x does not appear free in A, is easy to understand. It means that x does not really appear as itself in A, it can only appear as a bound variable where its name is irrelevant, so no replacements will be done, nothing can go wrong.

The second condition, y does not appear free in Z, is also quite easy. We are just looking at a β-reducible expression that could be inside a much larger expression, and y could be bound by a λy in the larger expression. If we don't change our own y's name, the y in Z will change its meaning, it will become bound to a different variable.

The need to generate a variable that has never been seen again, n, is easily solved. Treat variables as two part objects, names with subscripts. Plain old n is really n_0. If we keep track of the largest subscript that has ever been used with each name (or just one maximum so far for all names). The new variable is only ever used as a direct replacement for an existing variable (n for y in the rule above). To rename y_{75} throughout a sub-formula, just uniformly update its subscript. If the highest subscript yet used for y is 143 then all y_{75}s in the subformula become y_{144}s instead of ns.

A lambda expression is in "ground form" if nothing of the form $(\lambda x.A\ B)$ appears anywhere within it. It means the calculation is over, there is nothing left to do.

Here is a sequence of values following a pattern:

N_0 is $\lambda f.\lambda x.x$
N_1 is $\lambda f.\lambda x.(f\ x)$
N_2 is $\lambda f.\lambda x.(f\ (f\ x))$
N_3 is $\lambda f.\lambda x.(f\ (f\ (f\ x)))$
N_4 is $\lambda f.\lambda x.(f\ (f\ (f\ (f\ x))))$
N_5 is $\lambda f.\lambda x.(f\ (f\ (f\ (f\ (f\ x)))))$
N_6 is $\lambda f.\lambda x.(f\ (f\ (f\ (f\ (f\ (f\ x))))))$
etc.

I was careful to say "is" rather than = or anything like that. Those are simply giving shorthand names for some lambda expressions. They should be treated as macros, no name may appear in its own definition.

N_k may be thought of informally as a function of two parameters f and x where the first parameter f is expected to be a function itself. On receiving f and x, N_k applies f to x k times and returns the final result.

Here is something to go with that sequence:

I is $\lambda n.\lambda g.\lambda y.((n\ g)\ (g\ y))$

What is $(I\ N_0)$?

$(I\ N_0)$ is $(\lambda n.\lambda g.\lambda y.((n\ g)\ (g\ y))\ \lambda f.\lambda x.x)$

$(\lambda n.\lambda g.\lambda y.((n\ g)\ (g\ y))\ \lambda f.\lambda x.x)$
$\beta\equiv$ $\lambda g.\lambda y.((\lambda f.\lambda x.x\ g)\ (g\ y))$

$\lambda g.\lambda y.((\lambda f.\lambda x.x\ g)\ (g\ y))$
$\beta\equiv$ $\lambda g.\lambda y.(\lambda x.x\ (g\ y))$

$\lambda g.\lambda y.(\lambda x.x\ (g\ y))$
$\beta\equiv$ $\lambda g.\lambda y.(g\ y)$
$\alpha\equiv$ $\lambda g.\lambda x.(g\ x)$
$\alpha\equiv$ $\lambda f.\lambda x.(f\ x)$
is N_1

What is $(I\ N_4)$?

$(I\ N_4)$ is $(\lambda n.\lambda g.\lambda y.((n\ g)\ (g\ y))\ \lambda f.\lambda x.(f\ (f\ (f\ (f\ x)))))$

$(\lambda n.\lambda g.\lambda y.((n\ g)\ (g\ y))\ \lambda f.\lambda x.(f\ (f\ (f\ (f\ x)))))$

$\beta \equiv$ $\lambda g.\lambda y.((\lambda f.\lambda x.(f\ (f\ (f\ (f\ x))))\ g)\ (g\ y))$

$\beta \equiv$ $\lambda g.\lambda y.((\lambda f.\lambda x.(f\ (f\ (f\ (f\ x))))\ g)\ (g\ y))$
$\beta \equiv$ $\lambda g.\lambda y.(\lambda x.(g\ (g\ (g\ (g\ x))))\ (g\ y))$

$\beta \equiv$ $\lambda g.\lambda y.(\lambda x.(g\ (g\ (g\ (g\ x))))\ (g\ y))$
$\beta \equiv$ $\lambda g.\lambda y.(g\ (g\ (g\ (g\ (g\ y)))))$
$\alpha \equiv$ $\lambda g.\lambda x.(g\ (g\ (g\ (g\ (g\ x)))))$
$\alpha \equiv$ $\lambda f.\lambda y.(f\ (f\ (f\ (f\ (f\ y)))))$
is N_5

Here is something else to try:

A is $\lambda z.(z\ I)$ which really should be written as
$\lambda z.(z\ \lambda n.\lambda g.\lambda y.((n\ g)\ (g\ y)))$

What is $((A\ N_3)\ N_4)$?

$((A\ N_3)\ N_4)$ is $((\lambda z.(z\ I)\ \lambda f.\lambda x.(f\ (f\ (f\ x))))\ N_4)$

$((\lambda z.(z\ I)\ \lambda f.\lambda x.(f\ (f\ (f\ x))))\ N_4)$
$\beta \equiv$ $((\lambda f.\lambda x.(f\ (f\ (f\ x)))\ I)\ N_4)$

$((\lambda f.\lambda x.(f\ (f\ (f\ x)))\ I)\ N_4)$
$\beta \equiv$ $(\lambda x.(I\ (I\ (I\ x)))\ N_4)$

$(\lambda x.(I\ (I\ (I\ x)))\ N_4)$
$\beta \equiv$ $(I\ (I\ (I\ N_4)))$

$(I\ (I\ (I\ N_4)))$
$\ldots \equiv$ $(I\ (I\ N_5))$
$\ldots \equiv$ $(I\ N_6)$
$\ldots \equiv$ N_7

now M is $\lambda x.\lambda y.((x\ (A\ y))\ N_0)$ which really should be written as
$\lambda x.\lambda y.((x\ (\lambda z.(z\ \lambda n.\lambda g.\lambda y.((n\ g)\ (g\ y)))\ y))\ \lambda f.\lambda x.x)$

What is $((M\ N_3)\ N_4)$?

$((M\ N_3)\ N_4)$ is $((\lambda x.\lambda y.((x\ (A\ y))\ N_0)\ N_3)\ N_4)$

$((\lambda x.\lambda y.((x\ (A\ y))\ N_0)\ N_3)\ N_4)$
$\beta \equiv$ $(\lambda y.((N_3\ (A\ y))\ N_0)\ N_4)$

$(\lambda y.((N_3\ (A\ y))\ N_0)\ N_4)$
$\beta \equiv$ $((N_3\ (A\ N_4))\ N_0)$

$((N_3\ (A\ N_4))\ N_0)$

is $((\lambda f.\lambda x.(f\ (f\ (f\ x)))\ (A\ N_4))\ N_0)$

$\beta\equiv$ $(\lambda x.((A\ N_4)\ ((A\ N_4)\ ((A\ N_4)\ x)))\ N_0)$

$(\lambda x.((A\ N_4)\ ((A\ N_4)\ ((A\ N_4)\ x)))\ N_0)$

$\beta\equiv$ $((A\ N_4)\ ((A\ N_4)\ ((A\ N_4)\ N_0)))$

$((A\ N_4)\ ((A\ N_4)\ ((A\ N_4)\ N_0)))$

$\ldots\equiv$ $((A\ N_4)\ ((A\ N_4)\ N_4))$

$\ldots\equiv$ $((A\ N_4)\ N_8)$

$\ldots\equiv$ N_{12}

S_1 is $\lambda x.\lambda y.x$

S_2 is $\lambda x.\lambda y.y$

$((S_1\ A)\ B)$ $\beta\beta\equiv$ A and

$((S_2\ A)\ B)$ $\beta\beta\equiv$ B no matter what A and B are

Pr is $\lambda x.\lambda y.\lambda s.((s\ x)\ y)$

Sel_1 is $\lambda p.(p\ S_1)$

Sel_2 is $\lambda p.(p\ S_2)$

if C is $((Pr\ M)\ N)$

$(Sel_1\ C)$ is $(Sel_1\ [(Pr\ M)\ N])$

is $(Sel_1\ [(\lambda x.\lambda y.\lambda s.((s\ x)\ y)\ M)\ N])$

is $(\lambda p.(p\ S_1)\ [(\lambda x.\lambda y.\lambda s.((s\ x)\ y)\ M)\ N])$

is $(\lambda p.(p\ \lambda x.\lambda y.x)\ [(\lambda x.\lambda y.\lambda s.((s\ x)\ y)\ M)\ N])$

$(\lambda p.(p\ \lambda x.\lambda y.x)\ [(\lambda x.\lambda y.\lambda s.((s\ x)\ y)\ M)\ N])$

$\beta\equiv$ $([(\lambda x.\lambda y.\lambda s.((s\ x)\ y)\ M)\ N]\ \lambda x.\lambda y.x)$

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

$\beta\equiv$ $([\lambda y.\lambda s.((s\ M)\ y)\ N]\ \lambda x.\lambda y.x)$

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

$\beta\equiv$ $(\lambda s.((s\ M)\ N)\ \lambda x.\lambda y.x)$

$(\lambda s.((s\ M)\ N)\ \lambda x.\lambda y.x)$

$\beta\equiv$ $((\lambda x.\lambda y.x\ M)\ N)$

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

$\beta\equiv$ $(\lambda y.M\ N)$

$(\lambda y.M\ N)$

$\beta\equiv$ M

Pr is the constructor for a linked list link, and Sel_1 and Sel_2 extract the head and tail (or car and cdr) from a link.

Swap is $\lambda x.((Pr\ (Sel_2\ x))\ (Sel_1\ x))$ which should be written as
$\lambda x.((\lambda x.\lambda y.\lambda s.((s\ x)\ y)\ (\lambda p.(p\ \lambda x.\lambda y.y)\ x))\ (\lambda p.(p\ \lambda x.\lambda y.x)\ x))$

if C is ((Pr M) N) again
(Swap C) is (Swap [(Pr M) N])
is $(\lambda x.[(Pr\ (Sel_2\ x))\ (Sel_1\ x)]\ [(Pr\ M)\ N])$

$(\lambda x.[(Pr\ (Sel_2\ x))\ (Sel_1\ x)]\ [(Pr\ M)\ N])$
$\beta \equiv$ $((Pr\ (Sel_2\ [(Pr\ M)\ N]))\ (Sel_1\ [(Pr\ M)\ N]))$

$((Pr\ (Sel_2\ [(Pr\ M)\ N]))\ (Sel_1\ [(Pr\ M)\ N]))$
$\ldots \equiv$ $((Pr\ N)\ (Sel_1\ [(Pr\ M)\ N]))$

$((Pr\ N)\ (Sel_1\ [(Pr\ M)\ N]))$
$\ldots \equiv$ $((Pr\ N)\ M)$

Step is $\lambda x.((Pr\ (I\ (Sel_1\ x)))\ (Sel_1\ x))$ which should be written as
$\lambda x.((\lambda x.\lambda y.\lambda s.((s\ x)\ y)\ (\lambda n.\lambda g.\lambda y.((n\ g)\ (g\ y))\ (\lambda p.(p\ \lambda x.\lambda y.x)\ x)))\ (\lambda p.(p\ \lambda x.\lambda y.x)\ x))$

What is (Step ((Pr N_0) N_0)) ?

$(Step\ ((Pr\ N_0)\ N_0))$
is $(\lambda x.([Pr\ (I\ (Sel_1\ x))]\ (Sel_1\ x))\ ((Pr\ N_0)\ N_0))$
$\beta \equiv$ $((Pr\ (I\ (Sel_1\ ((Pr\ N_0)\ N_0))))\ (Sel_1\ ((Pr\ N_0)\ N_0)))$

$((Pr\ (I\ (Sel_1\ ((Pr\ N_0)\ N_0))))\ (Sel_1\ ((Pr\ N_0)\ N_0)))$
$\ldots \equiv$ $((Pr\ (I\ N_0))\ (Sel_1\ ((Pr\ N_0)\ N_0)))$

$((Pr\ (I\ N_0))\ (Sel_1\ ((Pr\ N_0)\ N_0)))$
$\ldots \equiv$ $((Pr\ (I\ N_0))\ N_0)$

$((Pr\ (I\ N_0))\ N_0)$
$\ldots \equiv$ $((Pr\ N_1)\ N_0)$

and

$(Step\ ((Pr\ N_1)\ N_0))$
is $(\lambda x.([Pr\ (I\ (Sel_1\ x))]\ (Sel_1\ x))\ ((Pr\ N_1)\ N_0))$
$\beta \equiv$ $((Pr\ (I\ (Sel_1\ ((Pr\ N_1)\ N_0))))\ (Sel_1\ ((Pr\ N_1)\ N_0)))$

$((Pr\ (I\ (Sel_1\ ((Pr\ N_1)\ N_0))))\ (Sel_1\ ((Pr\ N_1)\ N_0)))$
$\ldots \equiv$ $((Pr\ (I\ N_1))\ (Sel_1\ ((Pr\ N_1)\ N_0)))$

$((Pr\ (I\ N_1))\ (Sel_1\ ((Pr\ N_1)\ N_0)))$

$$
\begin{array}{rl}
\ldots \equiv & ((\mathrm{Pr}\ (\mathrm{I}\ \mathrm{N}_1))\ \mathrm{N}_1) \\
\hline
 & ((\mathrm{Pr}\ (\mathrm{I}\ \mathrm{N}_1))\ \mathrm{N}_1) \\
\ldots \equiv & ((\mathrm{Pr}\ \mathrm{N}_2)\ \mathrm{N}_1)
\end{array}
$$

and

$$
\begin{array}{rl}
 & (\mathrm{Step}\ ((\mathrm{Pr}\ \mathrm{N}_8)\ \mathrm{N}_7)) \\
\text{is} & (\lambda x.([\mathrm{Pr}\ (\mathrm{I}\ (\mathrm{Sel}_1\ x))]\ (\mathrm{Sel}_1\ x))\ ((\mathrm{Pr}\ \mathrm{N}_8)\ \mathrm{N}_7)) \\
\beta \equiv & ((\mathrm{Pr}\ (\mathrm{I}\ (\mathrm{Sel}_1\ ((\mathrm{Pr}\ \mathrm{N}_8)\ \mathrm{N}_7)))\ (\mathrm{Sel}_1\ ((\mathrm{Pr}\ \mathrm{N}_8)\ \mathrm{N}_7))) \\
\hline
 & ((\mathrm{Pr}\ (\mathrm{I}\ (\mathrm{Sel}_1\ ((\mathrm{Pr}\ \mathrm{N}_8)\ \mathrm{N}_7))))\ (\mathrm{Sel}_1\ ((\mathrm{Pr}\ \mathrm{N}_8)\ \mathrm{N}_7))) \\
\ldots \equiv & ((\mathrm{Pr}\ (\mathrm{I}\ \mathrm{N}_8))\ (\mathrm{Sel}_1\ ((\mathrm{Pr}\ \mathrm{N}_8)\ \mathrm{N}_7))) \\
\hline
 & ((\mathrm{Pr}\ (\mathrm{I}\ \mathrm{N}_8))\ (\mathrm{Sel}_1\ ((\mathrm{Pr}\ \mathrm{N}_8)\ \mathrm{N}_7))) \\
\ldots \equiv & ((\mathrm{Pr}\ (\mathrm{I}\ \mathrm{N}_8))\ \mathrm{N}_8) \\
\hline
 & ((\mathrm{Pr}\ (\mathrm{I}\ \mathrm{N}_8))\ \mathrm{N}_8) \\
\ldots \equiv & ((\mathrm{Pr}\ \mathrm{N}_9)\ \mathrm{N}_8)
\end{array}
$$

now

$$
\begin{array}{rl}
 & ((\mathrm{N}_5\ \mathrm{Step})\ ((\mathrm{Pr}\ \mathrm{N}_0)\ \mathrm{N}_0)) \\
\text{is} & ((\lambda f.\lambda x.[f\ (f\ (f\ (f\ (f\ x))))]\ \mathrm{Step})\ ((\mathrm{Pr}\ \mathrm{N}_0)\ \mathrm{N}_0)) \\
\beta \equiv & (\lambda x.[\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ x))))]\ ((\mathrm{Pr}\ \mathrm{N}_0)\ \mathrm{N}_0)) \\
\hline
 & (\lambda x.[\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ x))))]\ ((\mathrm{Pr}\ \mathrm{N}_0)\ \mathrm{N}_0)) \\
\beta \equiv & (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ ((\mathrm{Pr}\ \mathrm{N}_0)\ \mathrm{N}_0))))))
\end{array}
$$

$$
\begin{array}{rl}
 & (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ ((\mathrm{Pr}\ \mathrm{N}_0)\ \mathrm{N}_0))))))) \\
\ldots \equiv & (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ ((\mathrm{Pr}\ \mathrm{N}_1)\ \mathrm{N}_0)))))
\end{array}
$$

$$
\begin{array}{rl}
 & (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ ((\mathrm{Pr}\ \mathrm{N}_1)\ \mathrm{N}_0))))) \\
\ldots \equiv & (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ ((\mathrm{Pr}\ \mathrm{N}_2)\ \mathrm{N}_1))))
\end{array}
$$

$$
\begin{array}{rl}
 & (\mathrm{Step}\ (\mathrm{Step}\ (\mathrm{Step}\ ((\mathrm{Pr}\ \mathrm{N}_2)\ \mathrm{N}_1)))) \\
\ldots \equiv & (\mathrm{Step}\ (\mathrm{Step}\ ((\mathrm{Pr}\ \mathrm{N}_3)\ \mathrm{N}_2)))
\end{array}
$$

$$
\begin{array}{rl}
 & (\mathrm{Step}\ (\mathrm{Step}\ ((\mathrm{Pr}\ \mathrm{N}_3)\ \mathrm{N}_2))) \\
\ldots \equiv & (\mathrm{Step}\ ((\mathrm{Pr}\ \mathrm{N}_4)\ \mathrm{N}_3))
\end{array}
$$

$$
\begin{array}{rl}
 & (\mathrm{Step}\ ((\mathrm{Pr}\ \mathrm{N}_3)\ \mathrm{N}_2)) \\
\ldots \equiv & ((\mathrm{Pr}\ \mathrm{N}_5)\ \mathrm{N}_4)
\end{array}
$$

D is $\lambda x.(Sel_2\ [(x\ Step)\ ((Pr\ N_0)\ N_0)])$

$$
\begin{array}{rl}
 & (D\ N_5) \\
\text{is} & (\lambda x.(Sel_2\ [(x\ Step)\ ((Pr\ N_0)\ N_0)])\ N_5) \\
\beta\equiv & (Sel_2\ [(N_5\ Step)\ ((Pr\ N_0)\ N_0)])
\end{array}
$$

$$
\begin{array}{rl}
 & (Sel_2\ [(N_5\ Step)\ ((Pr\ N_0)\ N_0)]) \\
\ldots\equiv & (Sel_2\ ((Pr\ N_5)\ N_4)) \\
\beta\equiv & N_4
\end{array}
$$

Given that, subtraction of Y from X is just ((y D) x), so

S is $\lambda x.\lambda y.((y\ D)\ x)$

$$
\begin{array}{rl}
 & ((S\ N_{10})\ N_4) \\
\text{is} & ((\lambda x.\lambda y.((y\ D)\ x)\ N_{10})\ N_4) \\
\beta\equiv & (\lambda y.((y\ D)\ N_{10})\ N_4)
\end{array}
$$

$$
\begin{array}{rl}
 & (\lambda y.((y\ D)\ N_{10})\ N_4) \\
\beta\equiv & ((N_4\ D)\ N_{10}) \\
\text{is} & ((\lambda f.\lambda x.(f\ (f\ (f\ (f\ x))))\ D)\ N_{10})
\end{array}
$$

$$
\begin{array}{rl}
 & ((\lambda f.\lambda x.(f\ (f\ (f\ (f\ x))))\ D)\ N_{10}) \\
\beta\equiv & (\lambda x.(D\ (D\ (D\ (D\ x))))\ N_{10})
\end{array}
$$

$$
\begin{array}{rl}
 & (\lambda x.(D\ (D\ (D\ (D\ x))))\ N_{10}) \\
\beta\equiv & (D\ (D\ (D\ (D\ N_{10})))) \\
\ldots\equiv & (D\ (D\ (D\ N_9))) \\
\ldots\equiv & (D\ (D\ N_8))) \\
\ldots\equiv & (D\ N_7)) \\
\ldots\equiv & N_6
\end{array}
$$

T is S_1 which is $\lambda x.\lambda y.x$
F is S_2 which is $\lambda x.\lambda y.y$
If is $\lambda x.x$

If really isn't needed. All it does is make things look right.

$$
\begin{array}{rl}
 & (((If\ T)\ X)\ Y) \\
\text{is} & (((\lambda x.x\ \lambda x.\lambda y.x)\ X)\ Y) \\
\beta\equiv & ((\lambda x.\lambda y.x\ X)\ Y)
\end{array}
$$

$$
\begin{array}{rl}
 & ((\lambda x.\lambda y.x\ X)\ Y) \\
\beta\equiv & (\lambda y.X\ Y)
\end{array}
$$

$$
\begin{array}{rl}
 & (\lambda y.X\ Y) \\
\beta\equiv & X
\end{array}
$$

and

(((If F) X) Y)
is (((λx.x λx.λy.y) X) Y)
β≡ ((λx.λy.y X) Y)

((λx.λy.y X) Y)
β≡ (λy.y Y)

(λy.y Y)
β≡ Y

A test for zero, Z is λn.((n λy.F) T)

(Z N_0)
is (λn.((n λy.F) T) N_0)
β≡ ((N_0 λy.F) T)

---

is ((λf.λx.x λy.F) T)
β≡ (λx.x T)

(λx.x T)
β≡ T

and

(Z N_1)
is (λn.((n λy.F) T) N_1)
β≡ ((N_1 λy.F) T)

---

is ((λf.λx.(f x) λy.F) T)
β≡ (λx.(λy.F x) T)

(λx.(λy.F x) T)
β≡ (λy.F T)

(λy.F T)
β≡ F

and

(Z N_5)
is (λn.((n λy.F) T) N_5)
β≡ ((N_5 λy.F) T)

---

is ((λf.λx.(f (f (f (f (f x))))) λy.F) T)
β≡ (λx.(λy.F (λy.F (λy.F (λy.F (λy.F x))))) T)

(λx.(λy.F (λy.F (λy.F (λy.F (λy.F x))))) T)
β≡ (λy.F (λy.F (λy.F (λy.F (λy.F T)))))
β≡ (λy.F (λy.F (λy.F (λy.F F))))

$\beta \equiv$ $(\lambda y.F\ (\lambda y.F\ (\lambda y.F\ F)))$
$\beta \equiv$ $(\lambda y.F\ (\lambda y.F\ F))$
$\beta \equiv$ $(\lambda y.F\ F)$
$\beta \equiv$ F

And is $\lambda x.\lambda y.((x\ y)\ F)$
Or is $\lambda x.\lambda y.((x\ T)\ y)$
Not is $\lambda x.((x\ F)\ T)$

Equal x y	should be	(x - y is zero) and (y - x is zero) so
NotEqual x y	should be	(x - y is not zero) or (y - x is not zero) so
Less x y	should be	(x - y is zero) and (y - x is not zero)
LessOrEqual x y	is just	(x - y is zero)

Equal is $\lambda x.\lambda y.((And\ (Z\ ((S\ x)\ y)))\ (Z\ ((S\ y)\ x)))$
Neq is $\lambda x.\lambda y.((Or\ (Not\ (Z\ ((S\ x)\ y))))\ (Not\ (Z\ ((S\ y)\ x))))$
Less is $\lambda x.\lambda y.((And\ (Z\ ((S\ x)\ y)))\ (Not\ (Z\ ((S\ y)\ x))))$
Leq is $\lambda x.\lambda y.(Z\ ((S\ x)\ y)))$

Or we could make a separate NotZero:

Z is $\lambda n.((n\ \lambda y.F)\ T)$
NZ is $\lambda n.((n\ \lambda y.T)\ F)$

The Paradoxical Combinator

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

What does it do? Let's find out:

$(Y\ Q)$
is $(\lambda f.(\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x)))\ Q)$
$\beta \equiv$ $(\lambda f.(f\ (\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x))))\ Q)$

$(\lambda f.(f\ (\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x))))\ Q)$
$\beta \equiv$ $(\lambda f.(f\ (f\ (\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x)))))\ Q)$

$(\lambda f.(f\ (f\ (\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x)))))\ Q)$
$\beta \equiv$ $(\lambda f.(f\ (f\ (f\ (\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x))))))\ Q)$

$(\lambda f.(f\ (f\ (f\ (\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x)))))))\ Q)$
$\beta \equiv$ $(\lambda f.(f\ (f\ (f\ (f\ (\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x))))))))\ Q)$

$(\lambda f.(f\ (f\ (f\ (f\ (\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x))))))))\ Q)$
$\beta \equiv$ $(\lambda f.(f\ (f\ (f\ (f\ (f\ (\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x)))))))))\ Q)$

This is going nowhere. Try again with a different choice for the first reduction

$$
\begin{array}{rl}
 & (Y\ Q) \\
\text{is} & (\lambda f.(\lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x)))\ Q) \\
\beta\equiv & (\lambda x.(Q\ (x\ x))\ \lambda x.(Q\ (x\ x))) \\
\hline
 & (\lambda x.(Q\ (x\ x))\ \lambda x.(Q\ (x\ x))) \\
\beta\equiv & (Q\ (\lambda x.(Q\ (x\ x))\ \lambda x.(Q\ (x\ x)))) \\
\hline
 & (Q\ (\lambda x.(Q\ (x\ x))\ \lambda x.(Q\ (x\ x)))) \\
\beta\equiv & (Q\ (Q\ (\lambda x.(Q\ (x\ x))\ \lambda x.(Q\ (x\ x))))) \\
\hline
 & (Q\ (Q\ (\lambda x.(Q\ (x\ x))\ \lambda x.(Q\ (x\ x))))) \\
\beta\equiv & (Q\ (Q\ (Q\ (\lambda x.(Q\ (x\ x))\ \lambda x.(Q\ (x\ x)))))) \\
\hline
 & (Q\ (Q\ (Q\ (\lambda x.(Q\ (x\ x))\ \lambda x.(Q\ (x\ x)))))) \\
\beta\equiv & (Q\ (Q\ (Q\ (Q\ (\lambda x.(Q\ (x\ x))\ \lambda x.(Q\ (x\ x))))))) \\
\hline
 & (Q\ (Q\ (Q\ (Q\ (\lambda x.(Q\ (x\ x))\ \lambda x.(Q\ (x\ x)))))) \\
\beta\equiv & (Q\ (Q\ (Q\ (Q\ (Q\ (\lambda x.(Q\ (x\ x))\ \lambda x.(Q\ (x\ x)))))))
\end{array}
$$

and so on. It doesn't seem much better. But...

$$(Y\ Q)\ \beta\equiv\ (Q\ (Y\ Q))$$

That is not the definition of Y, just a fact about it. Y is not recursive.

Impf is $\lambda f.\lambda n.([(If\ (Z\ n))\ N_1]\ [(M\ n)\ (f\ (D\ n))])$

To make the intent clear, this means that

(Impf f) behaves as $\lambda n.$ if (n == 0)
the result is 1
else
the result is n * f (n - 1)

Note that there is no recursion in there.

Consider (worthless x) =
if (x == 3) return 6;
else if (x == 4) return 17;
else if (x == 5) return 120;
else if (x == 7) return 5040;
else return 2024;

worthless is a very poor approximation to factorial, it only gets it right for 3, 5, and 7.

What about (Impf worthless)?

```
(Impf  worthless) is      λn.  if (n == 0)
                                  the result is 1
                               else
                                  the result is n * (worthless  (n - 1))
```

((Impf worthless) 0) = 1, ✓
((Impf worthless) 1) = 2024,
((Impf worthless) 2) = 4048,
((Impf worthless) 3) = 6072,
((Impf worthless) 4) = 24, ✓
((Impf worthless) 5) = 85,
((Impf worthless) 6) = 720, ✓
((Impf worthless) 7) = 14168,
((Impf worthless) 8) = 40320, ✓
((Impf worthless) 9) = 18216,
((Impf worthless) 10) = 20240. It was correct four times.

((Impf (Impf worthless)) 0) = 1, ✓
((Impf (Impf worthless)) 1) = 1, ✓
((Impf (Impf worthless)) 2) = 4048,
((Impf (Impf worthless)) 3) = 12144,
((Impf (Impf worthless)) 4) = 24288,
((Impf (Impf worthless)) 5) = 120, ✓
((Impf (Impf worthless)) 6) = 510,
((Impf (Impf worthless)) 7) = 5040, ✓
((Impf (Impf worthless)) 8) = 113344,
((Impf (Impf worthless)) 9) = 362880, ✓
((Impf (Impf worthless)) 10) = 182160. It was correct five times.
((Impf (Impf (Impf worthless))) 0) = 1, ✓
((Impf (Impf (Impf worthless))) 1) = 1, ✓
((Impf (Impf (Impf worthless))) 2) = 2, ✓
((Impf (Impf (Impf worthless))) 3) = 12144,
((Impf (Impf (Impf worthless))) 4) = 48576,
((Impf (Impf (Impf worthless))) 5) = 121440,
((Impf (Impf (Impf worthless))) 6) = 720, ✓
((Impf (Impf (Impf worthless))) 7) = 3570,
((Impf (Impf (Impf worthless))) 8) = 40320, ✓
((Impf (Impf (Impf worthless))) 9) = 1020096,
((Impf (Impf (Impf worthless))) 10) = 3628800. ✓ It was correct six times.

It becomes clear that any/all correct answers produced by the original worthless are useless. (Impf x) will always produce a correct result for 0, no matter what x is, and additional applications of Impf will always extend the correctness one step further.

An unlimited number of Impfs will correctly calculate any factorial regardless of which initial function they are improving. Even one that never produces any result at all will be good enough. It doesn't even need to be a function:

((Impf x) N_0)
((λf.λn.([(If (Z n)) N_1] [(M n) (f (D n))]) x) N_0)
is ((λf.λn.([(If (Z n)) N_1] [(M n) (f (D n))]) x) N_0)
β≡ (λn.([(If (Z n)) N_1] [(M n) (x (D n))]) N_0)

(λn.([(If (Z n)) N_1] [(M n) (x (D n))]) N_0)
β≡ ([(If (Z N_0)) N_1] [(M N_0) (x (D N_0))])

([(If (Z N_0)) N_1] [(M N_0) (x (D N_0))])
is ([(λx.x (Z N_0)) N_1] [(M N_0) (x (D N_0))])
β≡ ([(Z N_0) N_1] [(M N_0) (x (D N_0))])

([(Z N_0) N_1] [(M N_0) (x (D N_0))])
is ([(λn.((n λy.F) T) N_0) N_1] [(M N_0) (x (D N_0))])
β≡ ([((N0 λy.F) T) N_1] [(M N_0) (x (D N_0))])

([((N0 λy.F) T) N_1] [(M N_0) (x (D N_0))])
is ([((λf.λx.x λy.F) T) N_1] [(M N_0) (x (D N_0))])
β≡ ([(λx.x T) N_1] [(M N_0) (x (D N_0))])

([(λx.x T) N_1] [(M N_0) (x (D N_0))])
β≡ ([T N_1] [(M N_0) (x (D N_0))])

([T N_1] [(M N_0) (x (D N_0))])
is ([λx.λy.x N_1] [(M N_0) (x (D N_0))])
β≡ (λy.N_1 [(M N_0) (x (D N_0))])

(λy.N_1 [(M N_0) (x (D N_0))])
β≡ N_1

Factorial is (Y Impf)

There is no recursion anywhere, and certainly no loops.

The very important Church-Rosser theorems

1. If more than one β reduction is possible at any point, you will not end up with a different ground form depending on which you choose to perform.

2. If more than one β reduction is possible at any point and any choice will eventually lead to a ground form, then choosing the leftmost first will eventually lead to a ground form. Just to be clear, the leftmost is the one whose (λ appears first.

The clumsy-sounding language of theorem 1 is necessary. You can't say what it seems to be trying to say, that every choice will produce the same result. Some choices may not produce a result (ground form) at all, as we saw in the first attempt at (Y Q). But regardless of choice all ground forms that can be reached are the same.

The lambda calculus itself is clearly the basis for real functional programming. Theorem 2 is the basis for lazy evaluation. A functional language has lazy evaluation if it always chooses the leftmost possible reduction first.