

$\lambda x. x$

“the λ calculus can be called *the smallest universal programming language of the world*”

Raul Rojas - A Tutorial Introduction To Lambda Calculus

Developed by Alonzo Church in the 1930's, the λ calculus is a universal system for expressing any **computable function**.

The λ calculus is equivalent to Turing machines,
but is not concerned with the machine.

The λ calculus only considers how functions can be expressed and evaluated.

There are two main components in the λ calculus:

1. expressions or “functions”
2. variables or “names”

There are two keywords in the λ calculus:

λ .

All expressions in the λ calculus have the structure:

$$\lambda x . x$$

$\lambda x . x$

The **head** of the expression: $\lambda x .$

The **body** of the expression: x

$\lambda x . x$

There is one **variable**: x

Let's talk about **functions**

Given the **function**:

$$f(x) = x$$

Given the **function**:

$$f(x) = x$$

What's the **domain**?

Given the **function**:

$$f(x) = x$$

What's the **codomain**?

Domain = { set of inputs }



$f(x) = x$



Codomain = { set of outputs }

Given $f(x) = x$

$$f(1) = 1$$

$$f(2) = 2$$

...

Domain = $\{ 1, 2 \dots \}$

Codomain = $\{ 1, 2 \dots \}$

How can we express a **function** in the λ calculus?

$$f(x) = x$$

$$f(x) = x$$

$$\lambda x . x$$

$$f(x) = x$$

$$\lambda x . x$$

head

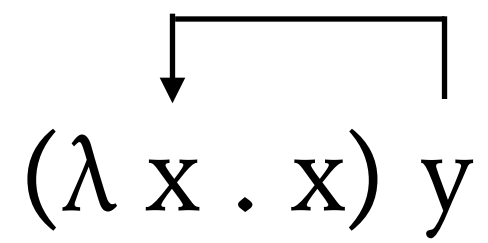
$$f(x) = \boxed{x}$$

$$\lambda x . \boxed{x}$$

body

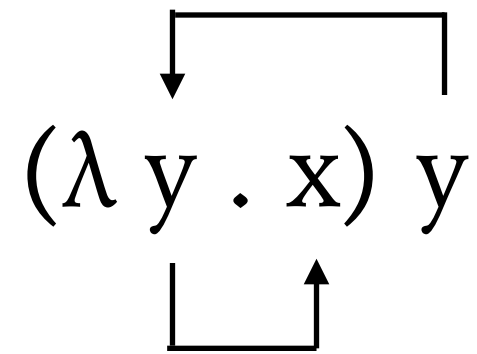
Let's talk about **applying expressions**

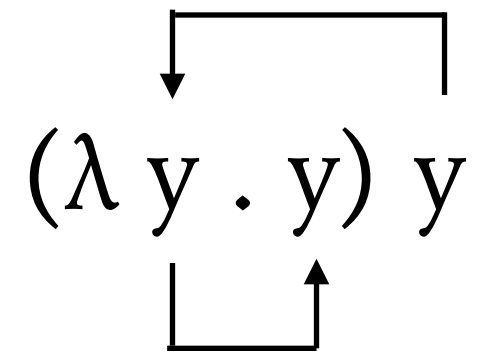
$(\lambda x . x) y$



The diagram shows the lambda expression $(\lambda x. x) y$. A horizontal line is positioned above the expression, starting from the space between the opening parenthesis and the variable x , and extending to the space between the closing parenthesis and the argument y . From the left end of this line, a vertical arrow points down to the variable x . From the right end of this line, a vertical line segment points down to the space between the closing parenthesis and the argument y .

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





y

Reduced form

$$(\lambda x . x) y = \textcolor{red}{y}$$

Redex (unreduced)

$(\lambda x . x) y$

Let's talk about **Alpha Equivalency**

Alpha Equivalency

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

Alpha Equivalency

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

Let's talk about **free** and **bound** variables

A free variable is not bound in the head

$(\lambda x . y)$

y is free

A **bound variable** is defined in the head

$(\lambda x . x)$

x is bound

$(\lambda x y . x z)$

What are the **bound variables**?

What are the **free variables**?

Let's talk about **Beta Reduction**

Beta Reduction is the **application** of expressions until no further **application** can occur:

$(\lambda x . x) y$

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

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

y

Let's try a slightly harder example:

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

Let's try a slightly harder example:

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

Let's try a slightly harder example still:

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

Let's try a slightly harder example still:

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

Expressions can be the **domain** of other expressions.

Expressions can be the **codomain** of other expressions.



Let's revisit this example:

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

Via **Beta Reduction**, we have shown that two expressions are **Alpha Equivalent**, meaning that they are the same expression.

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

We can also say that we have an expression, whose **domain** and **codomain** are always the same. This is known as the identity expression.

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

Let's talk about the **Y Combinator**

Given an expression Y , such that

$$Y \equiv (\lambda y . (\lambda x . y (x x)) (\lambda x . y (x x)))$$

Given an expression Y , such that

$$Y \equiv (\lambda y . (\lambda x . y (x x)) (\lambda x . y (x x)))$$


And given an expression R , such that


$$YR \equiv (\lambda y . (\lambda x . y (x x)) (\lambda x . y (x x))) R$$

Can we show that the **Y Combinator** allows us to employ recursion using nothing but **Beta Reduction** with lambda expressions (anonymous functions)?


Yes!


$$YR \equiv (\lambda y . (\lambda x . y (x x)) (\lambda x . y (x x))) R$$


$$YR \equiv (\lambda y . (\lambda x . y (x x)) (\lambda x . y (x x))) R$$


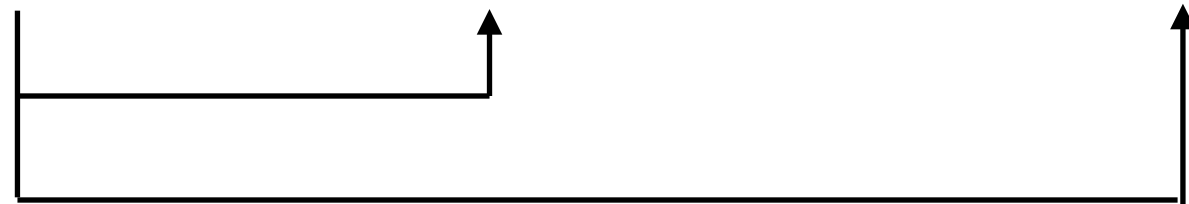
$$YR \equiv (\lambda y . (\lambda x . y (x x)) (\lambda x . y (x x))) R$$


$$YR \equiv (\lambda [y := R] . (\lambda x . y (x x)) (\lambda x . y (x x)))$$


$$YR \equiv (\lambda y . (\lambda x . y (x x)) (\lambda x . y (x x))) R$$



$$YR \equiv (\lambda [y := R] . (\lambda x . y (x x)) (\lambda x . y (x x)))$$


$$YR \equiv (\lambda y . (\lambda x . y (x x)) (\lambda x . y (x x))) R$$


$$YR \equiv (\lambda [y := R] . (\lambda x . y (x x)) (\lambda x . y (x x)))$$


$$YR \equiv (\lambda [y := R] . (\lambda x . R (x x)) (\lambda x . R (x x)))$$

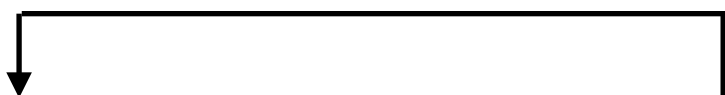
$$YR \equiv (\lambda y . (\lambda x . y (x x)) (\lambda x . y (x x))) R$$



$$YR \equiv (\lambda [y := R] . (\lambda x . y (x x)) (\lambda x . y (x x)))$$


$$YR \equiv (\lambda [y := R] . (\lambda x . R (x x)) (\lambda x . R (x x)))$$

$$YR \equiv (\lambda x . R (x x)) (\lambda x . R (x x))$$

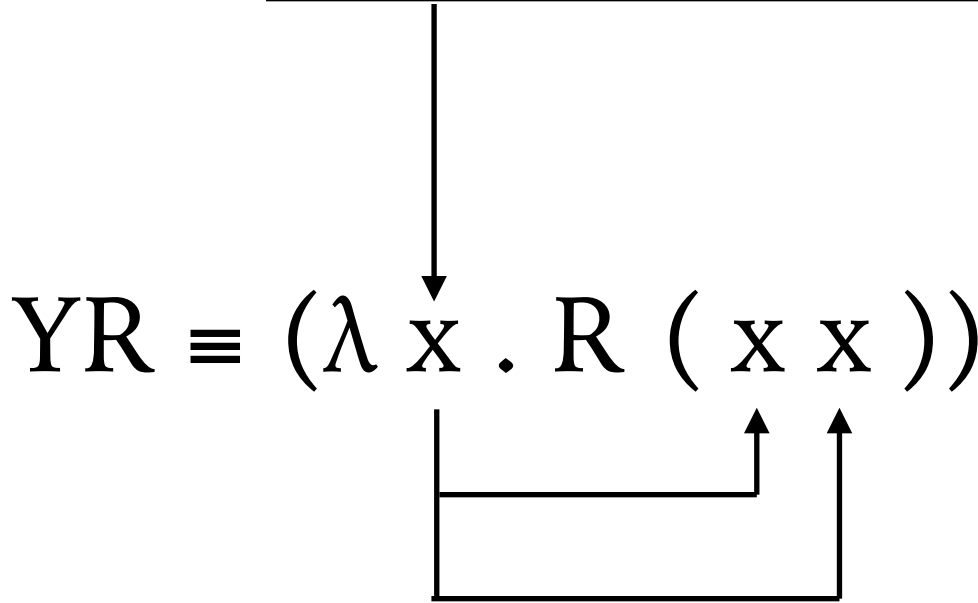
$$YR \equiv (\lambda x . R (x x)) (\lambda x . R (x x))$$

$$YR \equiv (\lambda x . R (x x)) \boxed{(\lambda x . R (x x))}$$


$$\boxed{(\lambda x . R (x x))}$$


$$YR \equiv (\lambda \dot{x} . R (x x))$$

$(\lambda x . R (x x))$



$$(\lambda x . R (x x))$$

$$YR \equiv (R (x x))$$

$$YR \equiv (R (\lambda x . R (x x)) (\lambda x . R (x x)))$$

What happens if we factor R?

$$YR \equiv (R (\lambda x . R (x x)) (\lambda x . R (x x)))$$

What happens if we factor R?

$$YR \equiv (R (\lambda x . R (x x)) (\lambda x . R (x x)))$$

$$Y \equiv (\lambda y (\lambda x . y (x x)) (\lambda x . y (x x)))R$$

We return to our original Y combinator form:

$$Y \equiv (\lambda y (\lambda x . \lambda y (x x))) (\lambda x . \lambda y (x x)) R$$



Why?



“From There To Here,
And Here To There,
Funny Things Are
Everywhere.”

Dr. Seuss



More resources:

Allen, Christopher and Moronuki, Julie. *Haskell Programming From First Principles*.
haskellbook.com

Rojas, Raul. *A Tutorial Introduction to the Lambda Calculus*. FU Berlin, WS-97/98.
<http://www.inf.fu-berlin.de/lehre/WS03/alpi/lambda.pdf>