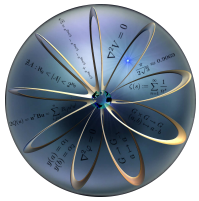

planetmath.org

Math for the people, by the people.

lambda calculus

Canonical name	LambdaCalculus
Date of creation	2013-03-22 12:32:39
Last modified on	2013-03-22 12:32:39
Owner	ratboy (4018)
Last modified by	ratboy (4018)
Numerical id	9
Author	ratboy (4018)
Entry type	Definition
Classification	msc 03B40
Related topic	CombinatoryLogic
Related topic	ChurchInteger
Related topic	RussellsParadox
Defines	pure lambda calculus
Defines	lambda abstraction
Defines	lambda expression

Lambda calculus (often referred to as λ-calculus) was invented in the 1930s by Alonzo Church, as a form of mathematical logic dealing primarly with functions and the application of functions to their arguments. In *pure lambda calculus*, there are no constants. Instead, there are only *lambda abstractions* (which are simply specifications of functions), variables, and applications of functions to functions. For instance, Church integers are used as a substitute for actual constants representing integers.

A lambda abstraction is typically specified using a *lambda expression*, which might look like the following.

$$\lambda\ x\ .\ f\ x$$

The above specifies a function of one argument, that can be *reduced* by applying the function f to its argument (function application is left-associative by default, and parentheses can be used to specify associativity).

The λ-calculus is equivalent to combinatory logic (though much more concise). Most functional programming languages are also equivalent to λ-calculus, to a degree (any imperative features in such languages are, of course, not equivalent).

Examples

We can specify the Church integer 3 in λ-calculus as

$$3 = \lambda f\ x\ .\ f\ (f\ (f\ x))$$

Suppose we have a function inc, which when given a string representing an integer, returns a new string representing the number following that integer. Then

$$3\ \text{inc}\ \texttt{"0"} = \texttt{"3"}$$

Addition of Church integers in λ-calculus is

$$\begin{aligned}
\text{add} &= \lambda\ x\ y\ .\ (\lambda\ f\ z\ .\ x\ f\ (y\ f\ z)) \\
\text{add}\ 2\ 3 &= \lambda\ f\ z\ .\ 2\ f\ (3\ f\ z) \\
&= \lambda\ f\ z\ .\ 2\ f\ (f\ (f\ (f\ z))) \\
&= \lambda\ f\ z\ .\ f\ (f\ (f\ (f\ (f\ z)))) \\
&= 5
\end{aligned}$$

Multiplication is

$$\begin{aligned}
\text{mul} &= \lambda\, x\, y\, .\, (\lambda\, f\, z\, .\, x\, (\lambda\, w\, .\, y\, f\, w)\, z) \\
\text{mul}\ 2\ 3 &= \lambda\, f\, z\, .\, 2\, (\lambda\, w\, .\, 3\, f\, w)\, z \\
&= \lambda\, f\, z\, .\, 2\, (\lambda\, w\, .\, f\, (f\, (f\, w)))\, z \\
&= \lambda\, f\, z\, .\, f\, (f\, (f\, (f\, (f\, z)))) \\
&= 6.
\end{aligned}$$

Russell's Paradox in λ-calculus

The λ-calculus readily admits Russell's Paradox. Let us define a function r that takes a function x as an argument, and is reduced to the application of the logical function not to the application of x to itself.

$$r = \lambda\, x\, .\, \text{not}\, (x\, x)$$

Now what happens when we apply r to itself?

$$\begin{aligned}
r\, r &= \text{not}\, (r\, r) \\
&= \text{not}\, (\text{not}\, (r\, r)) \\
&\vdots
\end{aligned}$$

Since we have not $(r\ r) = (r\ r)$, we have a paradox.