

The λ -calculus

Murdoch J. Gabbay, Heriot-Watt University, Scotland

*PGCAP presentation exercise, Heriot-Watt University,
Scotland
25 October 2006*

The λ -calculus

I am a theoretical computer scientist. So I need tools to study the **theory** of computing science! Here's one: the λ -calculus.

The λ -calculus is a framework for formally expressing and reasoning about computation. For example

$$\lambda x.(x + x)$$

means

'the function that takes x and returns $x + x$ '.

The λ -calculus

λ is not the name of a death-ray.

$$\lambda x.(x + x)$$

We could write

‘input x then output $x + x$ ’,

or

‘ $x + x$ with parameter x ’.

But that wouldn't be so sexy, would it?

The λ -calculus

Note that we do not have to say **how** we calculate $x + x$.

We could use

- a ruler and compass (greeks, 300BC),
- a slide rule (1900AD),
- or a computer chip (today).

We just specify **what** we calculate.

The λ -calculus is more abstract than Turing machines.

Functions

We call $\lambda x.x + x$ a **function**. This is mathematical terminology for something that takes an argument and gives a result. Another example of a function is a word processor, which may take a series of key strokes and produce a pdf file of slides for PGCAP, for example.

We usually write f for functions.

Obviously if $f = \lambda x.x + x$ we want to **apply** it to a number;

$f(2)$ is equal to 4 because

f is the function that takes a value x and returns $x + x$, and

$2 + 2$ is equal to 4.

λ -calculus function application

This is function application $f(x)$.

Theorem: $\lambda x.f(x)$ is always equal to f .

Proof:

f is the function that takes a value x and returns f applied to x .

$\lambda x.f(x)$ is the function that takes a value x and returns f applied to x .

These are equal!

λ -calculus code

This simple language is incredibly rich.

For example $\lambda f.\lambda x.f(f(x))$ represents the function:

Take a function f and take an argument x and return f applied **twice** to x .

It is possible to code numbers:

- 1 is $\lambda f.f$ (equal to $\lambda f.\lambda x.f(x)$ by our Theorem),
- 2 is $\lambda f.\lambda x.f(f(x))$,
- 3 is $\lambda f.\lambda x.f(f(f(x)))$,
- and so on — no need to **introduce** numbers.

λ -calculus code

Addition

$+$ is $\lambda f.\lambda g.\lambda h.\lambda x.f(h)(g(h)(x))$.

$+(1)(2)$ ('plus applied to 1, applied to 2') is equal to

$\lambda h.\lambda x.1(h)(2(h)(x))$ which is equal to **3**.

Conclusions

You can do a lot more than add numbers in the λ -calculus. In fact, you can do **anything**.

So to study computation it suffices to study the λ -calculus.

Computers — reduced to a **toy programming language!**