Lambda Calculus and its Impact on Computer Science

Joris Klaasse Bos
Zaanlands Lyceum

October 10, 2020

1 Prologue

The reason I chose this subject is that it combines two things I enjoy: abstract maths and computer science. I have been programming for about five to six years now. I mainly enjoy low-level programming, so naturally C is my most used language and I am most familiar with a simple procedural paradigm. Such a paradigm is, however, not always very easy to use when working with very large and complex systems. I, as many, started out with Object-Oriented Programming, but I did not like that very much. Therefore, I have been exploring alternative paradigms, including Data-Oriented Programming and Functional Programming. I am quite familiar with Data-Oriented Programming and the Rust programming language by now, but Functional Programming isn't something I ever really have got in to yet. I did find out about lambda calculus and combinatory logic, which intrigued me, but I haven't got into it beyond a basic level of understanding. That is why I decided to research it for this paper.

2 Introduction

Lambda calculus is, as its name suggests, a calculus. A calculus is a system of manipulating symbols, which by themselves don't have any semantic meaning, in a way that is somehow meaningful. We all know algebra. Algebra itself doesn't have an innate meaning, but we can use it to represent and solve real world problems. Algebra, however, is limited. Not every problem can be represented in algebra. There are many branches of mathematics that use different systems. One example would be formal logic, which is used for logical operations on booleans. Another such example is lambda calculus.

2.1 A short history

2.2 The syntax

Lambda calculus is all about unary anonymous functions. Such a function has no name to identify it, takes only one input, and returns a single expression that is only dependent on the input, so it doesn't have any outside state. A simple function definition in lambda calculus looks as follows:

$$\lambda a.a$$

The lambda signifies a function. Everything following it will be part of that function's definition. The a before the . is the name of the argument. There is only one, because, as I said before, all functions in lambda calculus are unary. Everything following the . is part of the return expression. The funcion above is the identity function in lambda calculus; it just returns the input. This is the equivalent of multiplying by one, or defining a function like $f(x) = x$, or multiplying a vector with the identity matrix; it does nothing.

But how do we use this function? Well, just like defining a function, it is quite simple. If you want to apply this function to a symbol, you just put it in parentheses behind the symbol. Something like this:

$$x(\lambda a.a)$$

Which evaluates to:

$$x$$

I have now basically explained the entire lambda calculus, it is really that simple. I have explained abstraction (the functions), application (applying a function to a symbol), and grouping (the parentheses), which is basically all we need. You can also give names to expressions. We could name our identity function I as follows:

$$I := \lambda a.a$$

But this isn't really part of the core lambda calculus anymore, just some syntactic sugar. This way, instead of constantly having to write $\lambda a.a$, we can just write I. So instead of writing:

$$x(\lambda a.a) = x$$

We could use our previous definition of I and write:

$$Ix = x$$

We have now covered identifiers too.

2.3 Combinatory logic