Lambda Calculus

ORMC - Advanced 1

Richemond Shin [1]

27 October 2024

1 Introduction

Lambda Calculus, often written as λ-calculus (where λ is the Greek letter "lambda"), is a system in mathematical logic and computer science used to describe how functions work. The idea is to only use one type of object, a function. Yet despite this, this approach allows us to use λ-calculus as a foundational approach to everything.

The core of λ-calculus is the λ-function. And to compute an expression involves the use of transformations and reductions which you will soon become acquainted with. Syntactically, the λ-function starts with a λ, then proceeded by the variable/input of our function, a dot, and then our body/output.

For example, $\lambda x.x + 1$ is the same as $f(x) = x + 1$. Dissecting our example:

1. λ - tells us our function definition has started (λ doesn't have any other special meaning outside of this).
2. x - our input variable of the function. We say that x is bound to the function.
3. . - this dot separates the declarations of our input and body
4. $x + 1$ - the output of our function.

Problem 1. "Translate" the following functions:

- $f(x) = x$
- $f(y) = y^2$
- $\lambda z.4z$

And like with the functions we're familiar with, we can also plug in expressions. For example, let's say we wanted to plug in 5 into a function $x^3 + 1$. Written in λ-calculus, $(\lambda x.x^3 + 1)5 = 5^3 + 1 = 125 + 1 = 126$. Note how when we plug in 5 into the equation, the λ and bound variable to the right of it disappears.

It's also important to clarify that we say 5 is the argument, and we apply our function to 5. It might not make sense how we "apply" a function onto a number, but we'll clarify that after the following problem.

[1] Based on a packet by Mark Ponomarenko

Problem 2. Compute the following:

- $(\lambda x.x)5$
- $(\lambda x.x^2)3$
- $(\lambda z.4z + y)4x$

In the previous problems, we've been working quite a lot with some arithmetic operations. But this is not entirely accurate. As said, there exists only one kind of object in λ-calculus: functions. And λ-calculus can only really transform or abstract these functions.

This means that operations such as addition or multiplication don't exist. And strictly speaking, when people use λ-calculus, they never reference numbers at all, only functions! So the previous problems have been somewhat misleading :(

If this sounds all confusing, do not worry as we'll be shifting towards a "purer" understanding of λ-calculus.

2 The Basics

So let's define some "pure" λ-functions:

- $I = \lambda x.x$
- $M = \lambda x.xx$

Both I and M take x as an input. I simply returns x. Meanwhile, M is a bit more interesting: it applies the function x on a copy of itself. Note how these λ-functions are only taking an expression and applying it to another (no fancy arithmetic here).

Also, note that M and I don't have a meaning on their own. They are not formal λ-functions. Rather, they are abbreviations that say "$\lambda x.x$ whenever you see I."

Problem 3. Simplify the following:

- $I\ I$
- $M\ I$
- $(I\ I)\ I$
- $\lambda a.(a\ (a\ a)))\ I$
- $((\lambda a.(\lambda b.a))\ M)\ I$

λ-functions are left-associative. So $(f\, g\, h) = ((f\, g)\, h)$. And expressed parentheses override this rule.

Problem 4. Remove as many parentheses as possible from the following expressions without changing their structure.

- $\lambda a.(a\,(a\,a)))\, I$
- $((\lambda a.(\lambda b.a))\, M)\, I$
- $((a\,b)(c\,d))((e\,f)(g\,h))$

As you may have noticed, sometimes a variables will not be defined as an input. Such as in the case of b in $\lambda a.b$, this variable is known as a free variable. The opposite of this, or when a variable is defined as an input, is known as a bound variable.

Problem 5. Determine the bound and free variables of each expression:
Hint: For "nested" expressions, it may help to work outside-in and step-step.

- $\lambda a.(a\,(a\,a)))$
- $((a\,b)(c\,d))((e\,f)(g\,h))$
- $(\lambda x.((\lambda y.(yx))(\lambda v.v)z)u)(\lambda w.w)$

The specific name of a bound variable does not affect the meaning of the expression. What this means is that for any expression with bound variables, we can replace them with any other variable, as long as it does not conflict with a free variable. So, $\lambda a.(az) = \lambda x.(xz)$ (note how this expression has the free variable z, so we avoid renaming our bound variable to z). This concept is known as α-equivalence, and we say that two λ-functions are α-equivalent if you can replace any expression for the bound variables, while not changing the free variables and keeping the structure of the functions the same.

Problem 6. One of the following expressions is not α-equivalent to $\lambda x.(\lambda x.x)$. Determine which one:

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

Problem 7. Let $K = \lambda a.(\lambda b.a)$. We can call K the "constant function function." Why does this name make sense?

Problem 8. Show that associativity matters by evaluating $((MK)I)$ and $(M(KI))$. What would MKI reduce to?

As seen in the previous problems, it is possible to "nest" expressions. Recall how the structure of a λ-function stipulates that only one input variable be used. So by chaining multiple together, we can "create" multivariate functions. This is known as *currying*.

Problem 9. Say $C = \lambda f.(\lambda g.(\lambda x.(f(g(x)))))$. What does C do? Evaluate $(C\ q\ r\ s)$ for some arbitrary expression q, r, and s.
Hint: Recall how λ-calculus is left-associative. f, g, and x are all bound variables in C

Problem 10. Evaluate $C\ M\ I\ a$. Then, evaluate $C\ I\ M\ I$.
Hint: a represents an arbitrary expression. Treat it like an unknown variable

Curried expressions can get quite long. So, we can use some shorthand to simplify the equations. Firstly by removing the unnecessary parentheses. And secondly by removing repeated occurrences of λ and dots. For example, we can simplify $\lambda x.(\lambda y.(\lambda z.((((x)x)y)z)))$ to $\lambda xyz.xxyz$.

Problem 11. Simplify C using this shorthand.
Hint: Think twice about removing certain parentheses!

Remember that this is only notation. Curried λ-functions are not multivariable functions, they are simply shorthand! Any λ-function presented with this notation must still be evaluated one variable at a time, just like an un-curried λ-function. Substituting all curried variables at once will cause errors.

As you hopefully have become quite acquainted with, it is possible manipulate, apply, and simplify equations. This general process is known as β-reduction. So, for example, $(\lambda x.x)y \rightarrow^{\beta} y$ or $(\lambda x.x)(\lambda y.y) \rightarrow^{\beta} (\lambda y.y)$.

Do note that β-reduction is just some fancy terminology. You've already β-reduced a lot λ-functions!

Problem 12. Let $Q = \lambda abc.b$. Reduce $(Q\ a\ c\ b)$.
Hint: the variables in the expression are different than those in Q.

Problem 13. Reduce $((\lambda a.a)\lambda bc.b)\ d\ \lambda eg.g$.

As mentioned previously, λ-calculus is made up of only the λ-function (and some other minor syntactical rules and properties). So as it stands, there are no numbers. But what happens when you try to evaluate numbers anyways?

Problem 14. Reduce $(\lambda yx.yx)\ 5\ 6$. Does your result mean anything?

Reducing the expression gives us (5 6), which means that we are applying 5 onto 6. But what does it mean to apply a numeral on another numeral? Does it mean to multiply them, add them? In fact, it doesn't mean anything since neither number is defined as a λ-function.

So to get (5 6) to mean anything, we need to first encode 5 and 6, or define them as λ-functions. In fact, we will be defining all natural numbers later on in the packet. We're also be gonna do the same with the major arithmetic operations as well, all from λ-functions. These basic functions reveal the malleability of λ-calculus, which is capable of expressing nearly any computation, making it Turing-complete.

To start, let's explore how this Turing-completeness allows us to define the essential logic of truth values — Boolean Algebra — entirely within λ-calculus.

3 Boolean Algebra

Boolean Algebra is the math of logic, being entirely composed of truth values. So let's define true and false in λ-calculus.

$$T = \lambda ab.a$$

$$F = \lambda ab.b$$

Problem 15. What do T and F do?
Hint: Think about it in terms of "choosing."

While this behavior seems arbitrary, observe what happens when we try to implement the following statement: ifthen $= \lambda bxy.bxy$.

Problem 16. Compute (ifthen $T\ M\ N$) and (ifthen $F\ M\ N$), where M and N are random variables. What does ifthen do?
Hint: Note that ifthen is one function.

Problem 17. Write a λ-function NOT so that $(NOT\ T) = F$ and $(NOT\ F) = T$.
Writing your function in any style, shorthand or not, is fine

Problem 18. Write a λ-function for AND and OR

4 Numbers

Since the only objects we have in λ-calculus are functions, it turns out to be more convenient to think of quantities as adverbs (once, twice, thrice,...) rather than nouns (one, two, three ...). We'll start with zero. If our numbers are once, twice, and thrice, it may make sense to make zero don't. So let's reuse our false, or F, λ-function and set it as $\bar{0}$.

$$\bar{0} = \lambda sz.z$$

Problem 19. Find a way to write 1, 2, and 3 with λ-calculus. We'll call these *Church numerals*. Try to generalize it to find the Church numeral of an natural number, n.
Hint: Observe our definition for zero. It takes s (a successor) and z (zero) as arguments. So does it make sense why our expression for zero returns only z and no s?

Problem 20. What is $(4\, I)\, a$, where a is a random expression?

Problem 21. What is $(3\, NOT\, T)$? How about $(8\, NOT\, F)$?

Problem 22. Create a successor operation so that $1 = S(0)$, $2 = S(1)$, and so on.
Hint: A good framework for this λ-function is λnsz, or more clearly $\lambda n.\lambda sz$. Do you see why?

Problem 23. Verify that $1 = S(0)$, $2 = S(1)$.

Problem 24. Define a λ-function ADD which adds two Church numerals.

Problem 25. Define a λ-function $MULT$ which multiplies two Church numerals.

Problem 26. Define a λ-function EXP which exponentiates a Church numeral to the power of another.

So far so good! Church numerals and basic arithmetic operations like the ones above are fairly straightforward to define. Unfortunately, an operation such as subtraction is surprisingly not. And even defining a predecessor λ-function, or a λ-function which increments down by one, is an ordeal in it of itself. But it is possible, requiring the use of ordered pairs. But we need to define them in the first place.

Problem 27. Design an expression $PAIR$ that constructs two-value tuples. For example, say $A = PAIR\ 1\ 2$. Then, $(A\ T)$ should reduce to 1 and $(A\ F)$ should reduce to 2.

For simplicity, you can write $(PAIR\ A\ B)$ as $\langle A, B \rangle$.

Problem 28. Write a λ-function H, which we'll call "shift and add." Given an input pair, it should shift its second argument left, then add one. So, $H\langle 0, 1\rangle$ should reduce to $\langle 1, 2\rangle$, $H\langle 1, 2\rangle$ to $\langle 2, 3\rangle$, and $H\langle 10, 4\rangle$ to $\langle 4, 5\rangle$.

Problem 29. Design a λ-function P that un-does S. That means $P(1) = 0$, $P(2) = 1$, etc. $P(0)$ should be zero.

Problem 30. Implement a λ-function SUB which subtracts a Church number from another.