Week 4 Notes (Lambda Calculus Applications and Meta-Logic)

February 20, 2024

1 Lecture Portion

1.1 Applications of Lambda Calculus

1.1.1 Boolean Logic

We can define true and false:

$$T = \lambda xy.x$$
$$F = \lambda xy.y$$

We can also define logical operators:

$$\wedge = \lambda xy.xyF$$
$$\vee = \lambda xy.xTy$$
$$\neg = \lambda x.xFT$$

Examples:

$$\wedge T\,T = T \tag{1}$$
$$\vee T F = T \tag{2}$$
$$\wedge(\vee F(\neg T))\,T = F \tag{3}$$

Exercise: Make sure these work.

1. $\wedge T\,T = T$

   $$(\lambda xy.xyF)(\lambda xy.x)(\lambda xy.x)$$

   Convert the symbols:

   $$(\lambda xy.xyF)(\lambda ab.a)(\lambda cd.c)$$
   $$(\lambda ab.a)(\lambda cd.c)F$$
   $$(\lambda cd.c) = T$$

2. $\vee T F = T$

   $$(\lambda xy.xFt)(\lambda ab.a)(\lambda cd.d)$$
   $$(\lambda ab.a)F(\lambda cd.d)$$
   $$(\lambda ab.a)(\lambda ef.f)(\lambda cd.d)$$
   $$(\lambda ef.f) = F$$

   Something must be wrong here...

3. $\wedge(\vee F(\neg T))\,T = F$

Another nice thing you can do is:

$$Q = \lambda bxy.bxy$$

which defines an operator that works with questions. That is to say, if the first argument is a boolean it will return the second argument on true and the third argument on false.

Examples:

$$QTcd = c$$
$$QFc(\lambda x.xx) = \lambda x.xx$$
$$Q(\wedge(\vee F(\neg T))T)cd = d$$

1.1.2 Church Numerals

We can define numerals like so:

$$0 = \lambda sz.z$$
$$1 = \lambda sz.s(z)$$
$$2 = \lambda sz.s(s(z))$$
$$3 = \lambda sz.s(s(s(z)))$$
$$4 = \lambda sz.s(s(s(z))))$$
$$\dots$$

Like with the booleans, these seem arbitrary but actually end up being excellent choices for definitions.

The symbols here form a mnemonic: s for successor, z for zero. Note that this is just a mnemonic. The letter z doesn't actually represent 0, that would be infinite self-reference!

We can define a sucessor function (add one):

$$S = \lambda wyx.y(wyx)$$

Exercise: Perform the following operations:

(Note: $\lambda wyx = \lambda w.\lambda y.\lambda x. = \lambda w.\lambda yx.$)

S0

$$(\lambda wyx.y(wyx))(\lambda sz.z)$$
$$(\lambda w.\lambda yx.y(wyx))(\lambda sz.z)$$
$$\lambda yx.y((\lambda sz.z)yx)$$
$$\lambda yx.y(x) = 1$$

S2

$$(\lambda wyx.y(wyx))(\lambda sz(s(s(z))))$$

Addition: + = $\lambda ab.aSb$($bearinmindthatnumbersarealsoiterators$!)

+23 = 2S3 = $(\lambda sz.s(s(z)))(\lambda wyx.y(wyx))(\lambda sz.s(s(s(z)))) = S4 = 5$

This constitutes a proof that 2+3=5. This is an example of the duality between programs and proofs.

Multiplication: * = $\lambda xyz.x(yz)Godel'stheoremappliestoeverylogicthatcanrepresentadditionandmultiplicati$

Pairs: represent an ordered pair (a, b)

pair = $\lambda abz.zab$

pair a b = $\lambda z.zab$

We can define functions to access elements first = $\lambda p.Tpsecond = \lambda p.Fp$

Predecessor function: (copy from notes. no intuitive way of explaining this)

Comparators:

is zero: Z = $\lambda n.nFTtestx >= yG = \lambda xy.Z(xPy)testx = yE = \lambda xy.AND(Gxy)(Gyx)(possiblemistake?)$

1.1.3 Recursion

Some expressions don't halt eg. $(\lambda x.xx)(\lambda x.xx)$

Consider: (stuff for Y combinator)

Y can be used to recurse functions of our choice.

(write out arithmetic sum function example)

YR5 = 15

This is the final part that make it Church complete.

1.2 Meta-Logic

Meta means something like study from outside, or an application of a subject to itself. From the greek word meaning after/beyond. Originally from Aristotle's metaphysics, which was just a sequel. (copy the list of meta things)

The MU system stuff we did was an example of metalogic.

Meta vs Strange Loops: Strange loops are the strongest form of meta. Meta is typically taking some of X to talk about some other part of X. Strange loops use some set of somethinng to talk about itself.

Church-Rosser Theorem: If an expression has a normal form then it doesn't matter what order you do reductions, you'll always arive at the normal form. This applies to lambda calculus, which means lambda calculus can be easily parallelised.

Halting Problem Proof:

[fill this in]

It's a function constructed specifically to halt if the input does not and vice-versa.

2 Tasks

2.1 Try some Boolean logic and Church arithmetic in a lambda interpreter and/or by hand.

2.2 Do the exercises in the lambda tutorial pdf.

2.3 Discuss the design of a lambda calculus interpreter:

2.3.1 eg. how does pylambda work?

2.3.2 eg. does pylambda has bugs thatyou could fix?

2.3.3 eg. design and build a better pylambda.

2.4 Can you convince yourself and others of the Halting problem proof?

2.5 What does the Halting problem mean for computation?