Combinators and Contradictions

April 23, 2023

When developing the fixedpoint theorem for LLLs Girard simply states

> “ *This is a straightforward imitation of Russell's paradox (already used in the fixpoint theorem of λ-calculus)* ”

Our goal here is to understand the relationship between the fixedpoint theorem of λ-calculus, Russell's paradox and the Y combinator.

Y Combinator

$$Y \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

is the Y combinator. Consider the following calculation

$$\begin{aligned} YX &= (\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))X \\ &=_\beta (\lambda x.X(xx))(\lambda x.X(xx)) \\ &=_\beta X((\lambda x.X(xx))(\lambda x.X(xx))) \\ &=_\beta X(YX) \end{aligned}$$

This is why Y is a fixed point combinator because it takes a λ term and returns the fixed point to this term.

Implementing Recursion

This term allows you to implement recursion into anonymous languages (such as λ-calculus) and for this reason can be used to prove the Turing completeness of the λ-calculus.

In a normal programing language we could impliment recursion by simply calling the name of the function being defined in the body of the definition, however anonymous functions do not get names and so we cannot do this. Instead we can give to Y a functional that represents what we are trying to define, where the recursive use of the function is now simply an argument. We can then observe that this function is what we were trying to define recursively but without explicit reference to itself.

The classic example is the factorial. Here is the standard sudo code for a factorial function:

```
define factorial(n):
  if n==0:
    return 1
  else:
    return factorial(n-1)

```

Notice that we call factorial by name within the definition. We could have defined:

```
define factorial_functional(f):
  if n==0:
    return 1
  else:
    return f(n-1)

```

Then we can observe that $new\_factorial \equiv Y(factorial\_functional)$ computes factorials as desired, without any explicit recursion.

I think that this doesnt work in practice becuase Y tries to compute every recursion at once, there is a different combinator Z that has a counter making this work on an actual computer I THINK..

Untypability

For the simply typed lambda calculus we have the rules

$$\Gamma, x : t \vdash x : t$$

$$\frac{\Gamma, x : t_1 \vdash M : t_2}{\Gamma \vdash \lambda x.M : t_1 \to t_2} \text{ Abst}$$

$$\frac{\Gamma \vdash M : t_1 \to t_2 \qquad \Gamma \vdash N : t_1}{\Gamma \vdash MN : t_2} \text{ App}$$

Y is a nontypable term in this system. This is essentially because

$$Y =_\beta \lambda f.(\lambda x.xx)(\lambda x.f(xx))$$

Consider the attempt to type Y and note that this essentially must be the last few steps of the typing process because of the almost deterministic typing rules.

$$\frac{\dfrac{f : t_1 \vdash \lambda x.xx : t_3 \to t_2 \qquad f : t_1 \vdash \lambda x.f(xx) : t_3}{f : t_1 \vdash (\lambda x.xx)(\lambda x.f(xx)) : t_2}}{\vdash \lambda f.(\lambda x.xx)(\lambda x.f(xx)) : t_1 \to t_2}$$

But this implies that $\lambda x.xx$ is typable which is not the case (contradicts strongly normalising).

Fixed Point Theorem for λ-Calculus

We say that a term, F, is a fixed point for another term, X, when $XF =_\beta F$

Theorem. *There is a fixed point for every λ-term*

Proof. Immediate because as we showed above for any term X

$$X(YX) = YX$$

.

Russell's Paradox

It is well known that it is possible to embed propositional/classical? logic into λ-calculus. For such a system we will define a $\neg$ term. By the fixed point theorem there is a fixed point for negation, i.e. a contradiction. Let us attempt to write down such a fixed point.

Inspired by Currys work on combinatorial logic we write down

$$\Omega = \lambda f.\neg ff$$

and then notice that trivially $\Omega\Omega = (\lambda f.\neg ff)\Omega = \neg\Omega\Omega$, thus a fixed point of negation. This can be called "Russell's paradox" simply because it contains the two key ideas of what has become a concept more than a particular paradox, namely self application and a fixed point for negation.

Notice that we did not actually use any construction for $\neg$ in λ-calculus and so this suggests a method of generating fixed points for any lambda term. If we wanted a fixed point for a term H then we should first define $\Gamma \equiv \lambda x.H(xx)$ then by our calculuation we see that $\Gamma\Gamma = H(\Gamma\Gamma)$. But we want a combinator for arbitrary H so we simply abstract again to consider $A = \lambda ax.a(xx)$ then we can calculate that

$$(AH)(AH) = ((\lambda ax.axx)H)((\lambda ax.axx)H) = (\lambda x.Hxx)(\lambda x.Hxx) = H((\lambda x.Hxx)(\lambda x.Hxx)) = H((AH)(AH))$$

So this is a functional that returns the analogous Ω term for arbitrary λ-term H. So we can now see that if we had a term $Z(X) = (AX)(AX) = \Omega_X\Omega_X$ then Z would be a fixedpoint combinator. To this end consider $Z = \lambda f.(Af)(Af)$. i.e.

$$Z = \lambda f.(Af)(Af) = \lambda f.(((\lambda ax.axx)f)((\lambda ax.axx)f)) = \lambda f.((\lambda x.fxx)(\lambda x.fxx)) = Y$$

So in fact what we have shown is that $Y(\neg) = \Omega\Omega$ from above. (Note that all equivalences are β)

Someone else has a nice explanation

http://c9x.me/notes/2015-06-02.html

Where xx in lambda calc translates to $x \in x$ in the set theory.

History

It appears that in A. Churches 1932 paper introducing the λ-calculus he actually addresses Russells paradox and introduces this construction. I am looking into this further out of interest.