The Fixed Point Theorem & Diagonal Argument

(after Tom Payne)

Recall that any λ-term f in the untyped λ-calculus has a fixed point: if

$$A = \quad x \longmapsto f(x(x))$$

then

$$A(A) = f(A(A))$$

so $A(A)$ is a fixed point of f. Tom Payne explained the magic behind this proof:

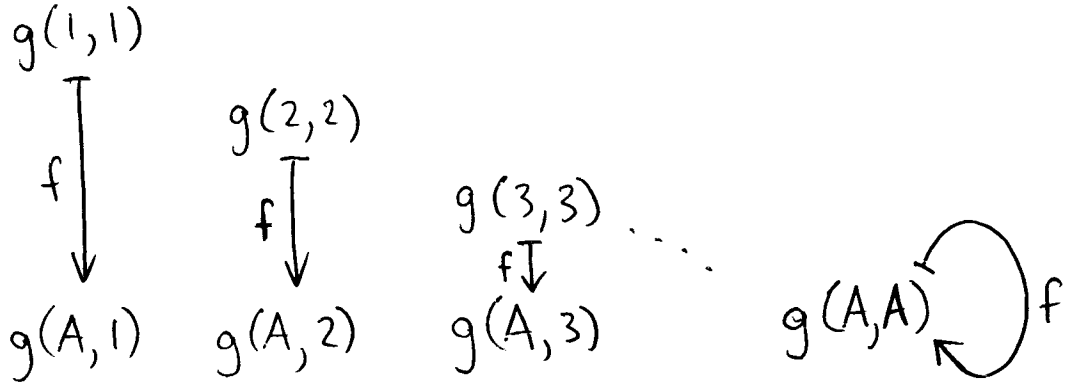
It's all about "diagonalization"!

Suppose you have a function g of two variables. You can draw it as a "matrix":

$$\begin{array}{ccccc} g(1,1) & g(1,2) & g(1,3) & \cdots \\ g(2,1) & g(2,2) & g(2,3) & \cdots \\ g(3,1) & g(3,2) & g(3,3) & \cdots \\ \vdots & \vdots & \vdots & \end{array}$$

Suppose there's a function of 1 variable f, which maps the diagonal entries to the entries of the Ath row:

$$f(g(x,x)) = g(A,x)$$



Tom imagines this as "scissors", &

$$f(g(A,A)) = g(A,A)$$

so $g(A,A)$ is a fixed point for f, the "hinge" of the scissors. This is a way to get fixed points! Alternatively, if we know f doesn't have a fixed point, we know that no such g, A can exist. Cantor used this in his famous diagonal argument.

Cantor

Suppose you could list all strings of bits (0's & 1's): ← infinite

	y=1	y=2	y=3		
x=1	0	0	0	0	0 ...
y=2	0	1	0	0	0 ...
	0	0	1	0	0 ...
	1	1	0	0	0 ...

⋮

Let $g(x,y)$ be the yth bit in the xth string.

Let f be

$$f(x) = \begin{cases} 0 & x = 1 \\ 1 & x = 0 \end{cases}$$

Then $f(g(x,x))$ is a string of bits which can't be on our list! Why? If it were the Ath string on our list:

$$f(g(x,x)) = g(A,x)$$

But taking $x = A$:

$$f(g(A,A)) = g(A,A)$$

so either

$$0 = 1$$

or

$$1 = 0$$ a contradiction.

Curry

Curry used the argument in a positive way to get a fixed point for any λ-term in the untyped λ-calculus. Now we're given a λ-term f and we want a fixed point. To get this, just find λ-terms g, A s.t.

$$f(g(x,x)) = g(A,x)$$

Then our fixed point will be $g(A,A)$. Since the λ-calculus is all about "application", let

$$g(x,y) = x(y)$$

Then we just need A s.t.

$$f(x(x)) = A(x)$$

What A will work?

$$A = x \longmapsto f(x(x))$$

This clearly works, so our fixed point for f is

$$g(A,A) = A(A)$$

So that's where the answer $A(A)$ came from.

2) Categorify everything.

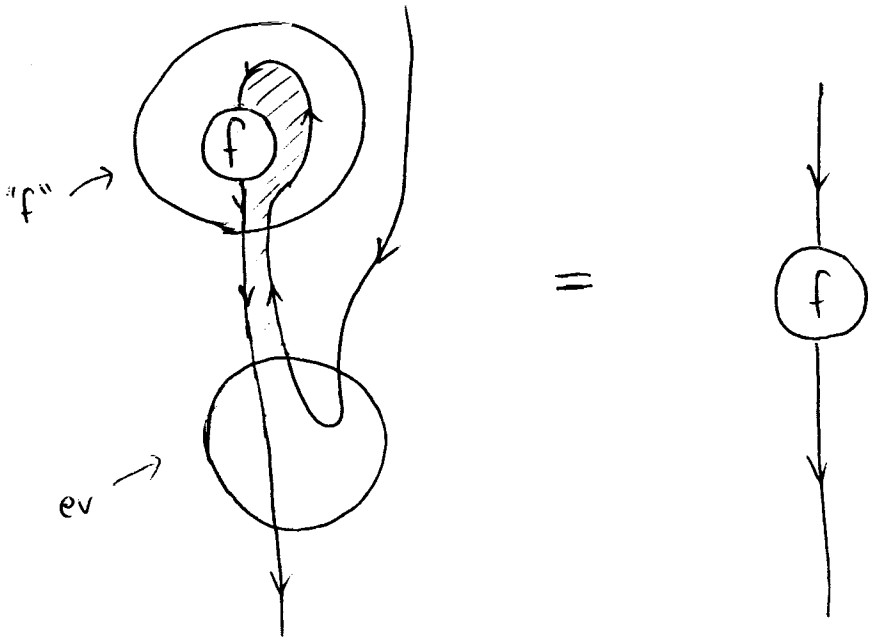
We've seen that using the λ-calculus, you can write programs to compute any computable function $f: \mathbb{N} \longrightarrow \mathbb{N}$. E.g. we had a homework problem to evaluate:

$$\Big\{\Big(\big[(f,x) \longmapsto f(f(f(x)))\big]\Big\{\big((g,y) \mapsto g(g(y))\big)\Big)(z \mapsto z+1)\Big\}(0)$$

We ground it out and got 8. By doing successive β-reductions, etc., we got a sequence of λ-terms starting with the above mess & ending with 8. All these terms were <u>equal</u> so we're missing out on the <u>process</u> of computation going from the first to the last λ-term. So we should replace equations between λ-terms by <u>morphisms</u>. That's categorification.

The λ-terms are morphisms in a CCC - so we need to introduce '2-morphisms' going between these morphisms. So we need "cartesian closed 2-categories", or "monoidal closed 2-categories" if we want to handle quantum computation.

E.g. in our string diagrams we had



We can draw this as a surface diagram: in the compact monoidal case

famous from singularity theory.