Solving Every Open Problem In Math With Category Theory

Max Vazquez

October 14, 2024

In this article, we take a very simple approach to solving every open problem in math, and we will talk about how category theory makes it easier and more efficient. As an introduction, let us say that there are two numbers ω, p such that $\omega + p \leq 6$, then one can easily compute $\omega^2 + p^2 \leq p$. This would be a non-trivial equation that cannot be reduced by category theory, but category theory allows us to build categories out of these equations:

$$\left(\begin{array}{c|cc} \Omega & \cdot & + \\ p^2 & - & 0 \end{array}\right)\left(\begin{array}{c|cc} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{array}\right) = \left(\begin{array}{c|cc} 3 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{array}\right).$$

A similar algorithm would need to be applied for other open problems (like finding an even number). It seems like a rather easy task, though, so let us make it more efficient first. We will introduce the concept of *iterated iterated iterated recursion*.

Firstly, to make things concrete, consider the iterated iterated recursion that is already there in calculus:

$$x = \frac{\sum_{i=1}^{k} x_i}{\sum_{j=1}^{m} y_j}$$

With x equal to the sum of its elements, we can use $\sum_i \sum_j x_i y_j$ to calculate x. It might be much more straightforward to compute x using our iterated iteration method, which yields:

$$x = x^2 + x + 2x^2 + \cdots.$$

We could also use another algorithm to find x:

$$x^n = \prod_{i=1}^{n} x_i.$$

For any $k > n$, we have that:

$$x^k = x^{k-1} \prod_{i=1}^{k-1} x_i^{k-1}.$$

If we used an iterated iterated iterated iteration method with the sum formula instead of multiplication, we would get a new x on the next pass, which is why category theory gives us such powerful tools.

Iterated iterated iterated recursion turns out to be a very powerful tool that does not require any additional calculations. Firstly, let us define it. Given two numbers x_0, x_1 and y_0, y_1, we write $xy = (x_0y_0) + (x_1y_1)$ for the product of their elements; we call them *times* of the same number. The definition follows the same construction as above. The function $xy = (x_0y_0) + (x_1y_1)$ then simply multiplies x_0y_0 by x_1y_1 and adds up all the results. We denote the final result as x. Now, given an iterated iterated iterated recursive function:

$$f(x_0, x_1, \ldots, x_n) = \{\, x_0\, if$$

$1 \text{ ¡ k} and f(x_0, \ldots, x_{k-1}) = f(x_{k-1}, \ldots, x_n)$
$x_n if 1 < k = n$.Then we may rewrite it in terms of times of numbers as:

$$x = \sum_{k=1}^{n} f(x_1, \ldots, x_n).$$

Then x is the unique number that computes $x = \sum_{k=1}^{n} f(x_1, \ldots, x_n)$. Similarly, we could consider the function $g(x_0, x_1, \ldots, x_n) = g(x_0, \ldots, x_{k-1})$. Then we have that:

$$g(x_0, \ldots, x_n) = \sum_{k=1}^{n} g(x_1, \ldots, x_k).$$

And finally, since both $f(x_0, x_1, \ldots, x_n)$ and $g(x_0, \ldots, x_{k-1})$ are functions of x, we can simplify the definitions of $xy = (x_0y_0) + (x_1y_1)$ and $g(x_0, \ldots, x_{k-1})$ in the way they did before.

$$xy = (x_0y_0) + (x_1y_1) + \sum_{k=1}^{n} f(x_1, \ldots, x_k) + \sum_{k=1}^{n} g(x_1, \ldots, x_k).$$

The name "iterated iterated iterated recursion" comes from the fact that the result of iterated iterated iterated iterated recursion is always the same as it gets computed again. We can think of xy as an example of this: xy is a product of a product of three products, while $g(x_0, \ldots, x_{k-1})$ is a product of a product of three products, while $f(x_0, x_1, \ldots, x_n)$ is a product of a product of three products, and so on.

We conclude the article by noting that many iterated iterated iterated recursive functions do not exist, but most of the iterated iterated iterated recursive functions form a group of the same type. However, even when $\omega^2 + \pm 3 = \pm 3\omega$, we cannot easily solve this equation using iterated iterated iterated iterated recursion because it involves computing $\omega^2 + \pm 3$ again. To remedy this, let us introduce a categorical approach.