You could have invented categories

- **Date**: 2019-11-19

After several years of trying my darndest to "get" category theory, I've had an epiphany. I think I "get" it now! But this is a double-edged sword: the basics of category theory seem so "intuitive" to me that I'm in danger of losing my ability to have a productive conversation with someone without that intuition.

While I still have both mindsets fresh, I want to explain my epiphany in the simplest terms I can. My approach to categories requires a little familiarity with some kind of axiomatic algebraic structure -- vector spaces are probably the most common structures for a CS undergrad, but they're way more complicated than we really need. Some familiarity with data structures may be valuable for the intuition, but it isn't obvious to me if that's necessary background.

If you know what a monoid is (**Not "monad"!** Don't freak out yet!), you're golden. Otherwise, the next section is just for you!

What's a "monoid"?

The word "monoid" may be unfamiliar, but things that are monoids *abound* in software. A monoid is any collection of things where (a) you can combine any two such things and (b) there's an "empty" thing that doesn't do anything when combined with something else.

- Strings are an easy monoid: you can concatenate them, and concatenating the empty string onto anything gives the same thing back. `"abc" + "def" = "abcdef"`; and `"abc" + "" = "abc" = "" + "abc"`.
- Integers can be combined using addition, and adding `0` to an integer gives the same thing back. `5 + 4 = 9`; and `5 + 0 = 5 = 0 + 5`.
- Integers can *also* be combined using multiplication, and multiplying `1` to an integer gives the same thing back. `5 * 4 = 20`; and `5 * 1 = 5 = 1 * 5`.
- Booleans can be combined with `AND`, with the empty element `true`.
- Booleans can *also* be combined with `OR`, with the empty element `false`.
- Square matrices can be combined using matrix multiplication, and multiplying a matrix by the identity matrix gives the same matrix back.
- Vectors (from a vector space) can be combined using addition, with the empty zero vector. (Vector spaces let us do more with vectors, but we just don't need any of that for a monoid!)

You might also combine binary trees, but they're *not* a monoid. Sure, given any two binary trees, you can stitch them together into a single binary tree with a new common root. But that kind of composition fails what's called "associativity": if I have a series of *three* binary trees, my final result depends on which order I stitch them together in. Do I stitch the first two together, and then add the third onto the right side? Or do I stitch the last

two together, and then add the first onto the left side? The answers are different!

For a more numerical example of non-associativity, think about subtraction. `(3 - 2) - 1` is very different from `3 - (2 - 1)`!

Requiring associativity amounts to working with list-like structures instead of tree-like structures [^o]. That's a simpler world, and we like simplicity, so we'll keep things associative.

[o]: In terms of trees, associativity means you can freely perform a tree rotation without changing the meaning of the expression. With enough rotations, you can effectively turn any tree into a linked list.

Monoids are great but...

Even though monoids show up all over the place, they have a pretty strict requirement: *everything* in the monoid has to be able to fit together. If I've got two numbers, then I can multiply them -- it doesn't matter in the slightest what the numbers actually are.

There are a lot of things in the world that just don't fit together so cleanly. Maybe you travel a lot, and you need a complete set of adapters between your devices and the various power standards around the world. Without those adapters, your equipment might get fried.

Or maybe you're like me, and you have a box of tangled-up cables in your closet. One day you get a new computer monitor, and to your *complete* dismay, your graphics card and your display have different interfaces. It sure would be nice to have a *single* standard for *transferring sequences* of data -- like some kind of Universal Serial Bus -- between your devices. But the world just isn't so perfect, so you have to go hunting for the right arrangement of cables.

Monoids sure seem nice, though. They're practically the essence of assembling a sequence of things together into a single gadget. Maybe we can tweak things a little bit.

What do clothes have to do with it?

If we're going to start shaking things up, we should probably simplify our scenario first. With regular monoids, if we had a value, we could stick another value onto either its left or its right side, and it could behave differently either way. (If our value is `"abc"`, we can concatenate something onto the left side, as in `"x" + "abc"`, or onto its right side, as in `"abc" + "x"`.) For now, let's pretend that doesn't happen -- that sticking something onto the left side is the same as sticking something onto the right side. We call this "commutativity", and it means there's really only one way to stick things together.

So we're in a commutative world right now, and we claim that there are some things that just can't be stuck together. How should we specify which things can get stuck together with which other things? I'm sure there are *plenty* of complicated things we could do, but let's go for the absolute dumbest, simplest thing possible: stick a little label onto it that warns us not to do something silly. (Like the "WASH COLD ONLY" clothes tags that everybody tears off without a second thought.)

Does this work? Actually, it does! [^1] It's just not very useful. Things with different tags can never interact at all -- we've essentially splintered our world into isolated little enclaves. We've got a constellation of independent monoids that we've put in the same room for no good reason, like high school cliques at the cafeteria.

Maybe this seems like a failure to you, but sticking labels on things *did* achieve the goal we set: incompatible things can't be fit together. It's just that we were a little heavy-handed. We could try more complicated labels and more subtle rules for sticking things together based on those labels, but it's starting to look more like guesswork down that road.

[1]: I'm lying, but only a little bit. You need an empty thing for each label; just one single empty thing for the whole monoid isn't enough anymore.

Nobody likes a long commute

Let's back up. Most things, like cables and strings, don't have just one side to stick things on. With our experience in the world of commutativity in mind, let's try to handle the situation where we can stick things onto either the left side or the right side of another thing. If we only needed one label when we only had one side... can we just give the left and right sides their own separate labels?

Sure can. That's what mathematicians (and Edward Kmett [^2]) call a category.

No, I'm serious, that's all there is to it. A category is a bunch of things, with rules for when you can stick them together. The things are two-sided, so you need to make sure that opposing sides are compatible. Compatibility is determined by a label attached to each side. The contents of the label don't really matter -- you just want a way to distinguish distinct interfaces.

Let's look back at matrices real quick. We talked about *square* matrices before, and they made a nice monoid, But what about rectangular matrices? Every matrix has a certain number of rows and a certain number of columns. You can only multiple two matrices if the number of rows in one matches the number of columns in another. Hey presto, a category!

Category theorists have built an incredible edifice of theory off of this *one* core idea. When you stick some adjectives before the word "category" -- like "monoidal category" or "Cartesian closed category" -- you're making your labels a little more interesting. You gain the ability to stick *labels* together, too, and they start to become their own little specification language for more and more complex and subtle ways of specifying when you can stick two things together. I don't claim that understanding vanilla categories trivializes the rest of the theory; but without this underlying principle, you couldn't even build the scaffolding to reach these concepts.

[2]: Edward Kmett is the author of the popular lens library for Haskell. He's *very* well versed in category theory.

This doesn't sound like a category

The standard nomenclature for what I've been calling a "label" is "object". I think this makes them appear much more important than they need to be for the basic idea to come across. Mathematicians do associate quite a lot of interesting and valuable information to objects, like sets and vector spaces and so on. But this is an *advanced play*.

What I've been calling, uh, "things", are normally called "arrows". Again, I think this makes them appear *less* important than they really are. A category with only one object is a monoid, which is a really useful thing in its own right. A category with no arrows just isn't useful at all on its own [^3]. Categories are all about composing things together... if you have no arrows, there's nothing to compose!

Emphasizing the arrows first -- the "things" that matter -- allows us to recognize objects ("labels") in the subordinate (but important!) role they play. When you stick two cables together, all that matters *from the outside* is the free ends that you can stick more cables onto. That doesn't mean that the actual assembly of cables doesn't have internal structure! It just means that the *labels* are only tracking the boundaries of your system.

One last thing before we end. If you're anything like me, you might be asking, "Hey, wait a moment -- a lot of things have more than two sides. What about that weird three-pronged Game Boy Advance link cable [^4] I never used because I didn't have any friends with a Game Boy?" Category theorists, the clever folks that they are, have figured out ways to track more than two sides using only two labels. A category like that requires a few extra bells and whistles in the same vein as the "more and more complex" labels I mentioned.

I'm not yet in a place where I can confidently pontificate on dealing with more than two labels, but my gut instinct is that it's similar to the situation in classical logic where "A and B entail C" is equivalent to "A entails (not B) or C". The word "entails" here is separating the left and right labels, and we're able to shuffle pieces of the label across in some fashion. If you can always expose just the interface you're trying to compose against on one side, then this lets you juggle multiple interfaces and stick things together only on the interface you immediately need to deal with.

[3]: They're useful when you also have another category handy that *does* have arrows, but let's not get into functors just yet!
[4]: This one, with two plugs and a socket.