Presheaves and Sheaves

Sam and Dennis

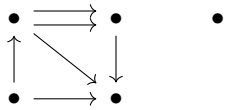
March 31, 2019

1 Graphs, again!

Recall a few lectures ago, we talked about directed graphs, and how they were actually examples of functors. We also talked about graph morphisms and how they were actually natural transformations. If you don't recall how that works, don't worry; we'll go over exactly how we got there today.

Let's recall what exactly a directed graph is: it consists of some set of vertices, and between any two vertices (say u and v), there can be some arrows (or edges) from u to v. In this case, we say the arrow has u as its source, and v as its target.

Here is an example:



There is another, more fun, graph. It is the graph of "love": we have one vertex for each human being. We have an arrow from one human to another iff the first human loves the second. This defines a very nice and important graph! Note, that depressingly, since we are only considering directed graphs, that there are times where you love someone else, but they don't love you back. There are also times where you, like that isolated point in the example, love no one. There are also cases where one person can love many different people, and where people can love themselves. What a cynical view of the world.

If one takes a step back, the examples look sort of like categories, don't they? In fact, they *are* categories! Except for one fatal flaw: they don't have composition. So, if you have an arrow from u to v and one from v to w, you *don't* necessarily have an arrow from u to w. Even if you do, the arrow from u to w has nothing to do with the two earlier arrows. Indeed, in general graphs, you have no way of combining two arrows into one, new arrow. This means that there are no identity arrows either (you need a composition to define this, since they have to be "identites with respect to" something!). Indeed, there may not even be any arrows from an object u to itself.

Notice that given any category, you can consider it a graph by forgetting about the composition and identities. If you consider $\mathbf{CAT}$ as the category of categories and functors between them (let's not worry about natural transformations right now), then this "forgetting" operation actually defines a functor from $\mathbf{CAT}$ to the category of graphs and graph morphisms, which we will denote by $\mathbf{Graph}$.

But what is a graph morphism? Let's recall those: a graph morphism f between two graphs G, G' firstly sends each vertex of G to a vertex of G', then each arrow of G to an arrow of

G' that preserves the source and target. In more "mathy" terms, given any arrow $e : u \to v$ in G (this means e has source u and target v), we have

$$f(s(e)) = s(f(e)), f(t(e)) = t(f(e)).$$

What this condition says is that things stick together correctly. It says that you cannot map the source of e in G to somewhere completely separate from where you map e. In fact, wherever you map e, you *must* map the source of e to the source of the arrow you mapped e to. This makes intuitive sense: then we see that there is a perhaps squashed image of the graph G inside G', which is given by f. This makes the intuition of "mapping G into G'" precise.

1.1 Two special buidling blocks

If you play around for a while, you can notice that every graph is in fact "built up" from two very simple graphs: the one-point graph and the one-arrow graph. We shall denote these graphs by P and A, where P is for point and A is for arrow.
Indeed, if you look at the example graph, call it G, given above, we see there is exactly one graph morphism from P to G for every vertex. Indeed the morphism only "points out" which object! Also, there is exactly one morphism from A to G for every arrow of G.
In a sense, graphs are defined by how P and A map into them! You can think of graphs as "spaces", all modelled off the two prototype "spaces", A and P. In fact, as mentioned above, for any general graph G we can get its set of vertices G_0 by looking at $\mathrm{Hom}_{\mathbf{Graph}}(P, G)$, the graph morphisms $P \to G$, and its set of arrows G_1 by looking at $\mathrm{Hom}_{\mathbf{Graph}}(A, G)$.
Let me now ask you a question. Say that I pick a graph G. Can you determine what graph it is, if I tell you how to map into it using A and P? Think about this for a while.
The answer is... yes and no. If by "telling you how to map into it" I mean that I only give you the two sets $\mathrm{Hom}_{\mathbf{Graph}}(P, G)$ and $\mathrm{Hom}_{\mathbf{Graph}}(A, G)$, then clearly not. I've basically told you how many vertices and how many arrows there are, but I never told you how they stick together! If I told you that G_0 is $\{0, 1, 2\}$ and that G_1 is $\{a, b, c\}$, you would still have no idea what graph I'm talking about, since you have no idea what the source and targets of a, b, c should be!
So, actually I cheated this game. I need to tell you something else about my graph, and it has to do with how the two graphs P and A map into each other. There are in fact two very special morphisms between P and A; can you think of what they are?
If you guessed the two vertices of A, you are right! As I said, morphisms from P to A must correspond with vertices of A! Let's call the map P to the source of the arrow s' and the map from P to the target of the arrow t', since they point out the source and target of the arrow in A respectively.
Now note that with these two graph morphisms, we can consider—instead of just P and A—the category with objects P, A and morphisms s' and t' (and the identity morphisms of P and A, but these are not so important).
Note that these two morphisms give us a connection between $\mathrm{Hom}_{\mathbf{Graph}}(A, G)$ and $\mathrm{Hom}_{\mathbf{Graph}}(P, G)$! Since for any morphism $a : A \to G$, we can precompose it with $s', t' : P \to G$ to give us $as', at' : P \to G$. This actually defines two maps $\mathrm{Hom}_{\mathbf{Graph}}(A, G) \to \mathrm{Hom}_{\mathbf{Graph}}(P, G)$, which we shall call s, t. Let's actually calculate what these two maps do: Given any map $a : A \to G$, it just point out some arrow (which we will also call a) of G. Now, notice that if we precompose with s' that $as' : P \to G$ now points out the source of a. Similarly,

precomposing with t' gives us the target of a. So the two maps from $\mathrm{Hom}_{\mathbf{Graph}}(A, G)$ and $\mathrm{Hom}_{\mathbf{Graph}}(P, G)$ are just the source and target maps!

Remember our game: I hid a graph G beind my back, and I promised to tell you how the graphs P and A map into it. If by this I mean that I will *also* tell you exactly the maps $A \to G$ interact with the maps $P \to G$ using the two maps $s', t' : P \to A$, then you *do* know what graph I'm talking about! This is because you have G_0, G_1 by the two sets I gave you, but you *also* have the source and target maps as they are given by how maps precompose with s', t'! So, you know how the edges glue together with the vertices.

In a sense, you can any identify graph G then with the above set of data: how P maps into it, how A maps into it, and how the two connect with s', t'. In fact, this data actually just gives a functor from the category of just P and A, given by

$$P \underset{t'}{\overset{s'}{\rightrightarrows}} A$$

into **Set**. Given any graph G it maps P to $\mathrm{Hom}_{\mathbf{Graph}}(P, G)$, maps A to $\mathrm{Hom}_{\mathbf{Graph}}(A, G)$. Furthermore on maps it sends s' to the map s and t' to the map t...wait, hold on. s', t' send $P \to A$, while s, t send $\mathrm{Hom}_{\mathbf{Graph}}(A, G) \to \mathrm{Hom}_{\mathbf{Graph}}(P, G)$! It's backwards! This is so strange, isn't it?

Actually, it isn't strange at all. Remember how we defined s, t: they were defined by *precomposition* by s', t'! Notice what happens during any precomposition: a map $f : A \to B$ determines a map from $\mathrm{Hom}(B, X) \to \mathrm{Hom}(A, X)$ since you *need* the domain to be B in order to *precompose* with f! With post-composition, this "switcheroo" doesn't happen. Please work this out on your own...

So, actually we have an "almost" functor from the category of P to A to **Set** that determines a graph. The only problem with this functor is that it switches the arrows! This is called a "contravariant" functor; a functor that switches arrows. Some sources call these functors "co-functors", but this terminology is *not* standard, so don't use it! Sometimes, to emphasize that a functor is a regular, non-switching functor, we shall say that it is covariant.

We can actually turn every contravariant functor into a covariant one, using the opposite category construction. Note that contravariant arrows switches all the arrows of its domain. So, we can switch the arrows too: call the opposite category of any category $\mathbf{C}$ the category whose objects are the same as those in $\mathbf{C}$, but whose arrows are switched from those of $\mathbf{C}$. (So a morphism $a \to b$ in $\mathbf{C}$ gives rise to an arrow $b \to a$ in $\mathbf{C}$). We denote this category by $\mathbf{C}^{\mathrm{op}}$.

So, every graph G determines and is determined by a contravariant functor $P \underset{t'}{\overset{s'}{\rightrightarrows}} A$ to **Set**, or if we denote the category $P \underset{t'}{\overset{s'}{\rightrightarrows}} A$ by **Proto**, each graph is determined by and determines a functor $\mathbf{Proto}^{\mathrm{op}} \to \mathbf{Set}$. (The determined by direction is simple: the functor specifies a vertex set and an arrow set, then specifies source and target. This is exactly how we defined directed graphs earlier! One can check that the two directions are inverse to each other).

This goes even further: any graph morphism actually determines a natural transformation of the functors, which we get using post-composition! As described, G is determined by $\mathrm{Hom}(-, G)$, a functor $\mathbf{Proto}^{\mathrm{op}} \to \mathbf{Set}$. Given any graph morphism $G \to H$, we get a natural transformation $\mathrm{Hom}(-, G) \to \mathrm{Hom}(-, H)$ given on components by $\mathrm{Hom}(P, G) \to \mathrm{Hom}(P, H)$ by post-composition by f, and $\mathrm{Hom}(A, G) \to \mathrm{Hom}(A, H)$ by post-composition

by f. Thus, we call this natural transformation $f_\star$, the "post-composition by f" transformation. Notice that since it is *post*-composition, the arrows do *not* switch this time!
CHECK ON THE BOARD THAT IT IS NATURAL, OR LEAVE IT AS EXERCISE
One can check that $f_\star$ gives a map $f_0 : G_0 \to H_0$ (given by its P component) and a map $f_1 : G_1 \to H_1$ (given by its A component) such that source and target is preserved. This turns out to just be a reformulation of the condition that $f_\star$ is natural! This also goes the other way: natural transformations $\mathrm{Hom}(A, G) \to \mathrm{Hom}(A, H)$ will unpack to maps $G_0 \to H_0$ and $G_1 \to H_1$ that preserve source and target, so gives a unique graph morphism $G \to H$. This is a consequence of the often stated "Yoneda Lemma", which we'll go over later.
So, we see that the whole category of graphs can be represented by the category whose objects are functors $\mathbf{Proto}^{\mathrm{op}} \to \mathbf{Set}$ and whose morphisms are natural transformations. We shall denote this "functor category" by $[\mathbf{Proto}^{\mathrm{op}}, \mathbf{Set}]$. Nicely, $\mathbf{Proto}$ embeds into this functor category by using the functor that sends $P \to \mathrm{Hom}(-, P)$, $A \to \mathrm{Hom}(-, A)$, etc. This is a consequence of the Yoneda lemma, which we shall see in more detail later. Actually, a lot of this example works because of the Yoneda lemma.
In this (very involved) example, we saw that in a very precise sense, we can say that graphs are modelled off the two "basic" graphs, P and A (and the two maps $s', t' : P \to A$, because $\mathbf{Graph}$ can be calculated as $[\mathbf{Proto}^{\mathrm{op}}, \mathbf{Set}]$.
In fact, we can do this in general. We start out with some "prototype spaces" that we want to build more general spaces out of; this forms a category $\mathbf{P}$ of prototypes (in this case it was the category with just A and P, the category $\mathbf{Proto}$). Then, we calculate the generalized spaces "built" from these prototypes as $[\mathbf{P}^{\mathrm{op}}, \mathbf{Set}]$, or some subcategory of this functor category (in particular, using some specified set of sheaves rather than all presheaves). In this sense, we can see $[\mathbf{P}^{\mathrm{op}}, \mathbf{Set}]$ as "generalized spaces". This is useful because many times there are "spaces" that we wish existed, for example limits or colimits, or classifying spaces, etc... but they do not exist in our prototype category, Instead, we extend our prototypes to all "spaces built from prototypes", and suddenly due to some categorical facts, everything nice finally exists.
One can now try some other simple categories $\mathbf{P}^{\mathrm{op}}$ and calculate just what $[\mathbf{P}^{\mathrm{op}}, \mathbf{Set}]$ should be. Whatever it is, the objects must be determined by how the objects of $\mathbf{P}$ map into it, where the objects $p \in \mathbf{P}$ are identified with their images $\mathrm{Hom}(-, p) \in [\mathbf{P}^{\mathrm{op}}, \mathbf{Set}]$. Or perhaps you'd rather wait for us to prove the Yoneda lemma before doing this...
Sheaves are just presheaves that satisfy some "descent" condition, which means that there is some nice gluing property that holds. Intuitively there is some nice way of specifying what counts as a "cover" as an object. Then, whatever presheaf you have should satisfy the descent property with respect to these covers, (that some gluing condition holds) to be a sheaf.