Category theory notes

January 21, 2019

They say you don't understand something until you can explain it, so here is my exercise in explaining category theory. I make no claims to any authority on this subject and the document is incomplete in proportion to my incomplete understanding.

1 Why

The first question I have had to explain, to myself and every single person I've spoken to about these things: why study category theory at all? This is forward looking, but here are my impressions halfway through a few textbooks:

- By *category*, we mean *context*, and we're looking for patterns across contexts. When two mathematical pursuits do the same thing, can we formalize exactly how they are or are not identical? There is a promise that even theoretical work in real-world contexts (i.e., my actual job) will benefit from better formalization of how contexts relate.
- I'm hoping it'll be a shortcut to learning more math. If I get this, then algebraic topology should be metaphorical cake.
- There is a trend in programming languages toward languages whose theory is based on a category of types. Out-of-fashion languages are written in terms of procedures: plug 4 into the square functions and we do some things and return 16. Languages oriented toward mathematical functions encourage a thinking that is far more still: the square of 4 is 16 and always has been and always will be.
- From what I've read, a lot of philosophers work on it—there's some presumption that it's not just about symbol shunting, but about finding 'natural' and 'universal' properties of those patterns across disparate things. The relationship between these categories/contexts has

always been there and we only need a way to see them. Here's Robert Penn Warren on the matter:

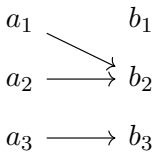
> Here is the shadow of truth, for only the shadow is true.
> And the line where the incoming swell from the sunset Pacific
> First leans and staggers to break will tell all you need to know
> About submarine geography, and your father's death rattle
> Provides all biographical data required for the *Who's Who* of the dead.
>
> ... In the distance, in *plaza, piazza, place, platz,* and square,
> Boot heels, like history being born, on cobbles bang.
>
> Everything seems an echo of something else.
>
> ... I watched the sheep huddling. Their eyes
> Stared into nothingness. In that mist-diffused light their eyes
> Were stupid and round like the eyes of fat fish in muddy water,
> Or of a scholar who has lost faith in his calling. ...
>
> You would think that nothing would ever again happen.
>
> That may be a way to love God.

OK, now that we've covered the *why* question, let's start from simple element-to-element mappings and work up.

2 Leveling up

A function $f : A \to B$, aka a mapping, assigns a value to all of the elements in the set A to some element(s) in the set B. Note that if $A = \{\emptyset\}$ then you still have a function, though it is vacuous.

Level 1: Objects=Elements We can draw this function via arrows, maybe

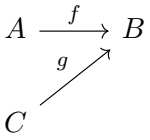


(Map)

Here, the nodes in the graph are single elements in the sets, and the arrows describe where each element goes.

Because of this multi-level issue, I'll mark what the arrows mean in any given diagram here with a side-note. In this case, the arrows are mappings, in the next they will be functions, and so on down the line.

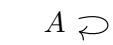
Level 2: Objects=Sets Level 1 showed elements internal to the sets, and at the next level we'll look externally, treating each set as single object. The function notation $f : A \to B$ is itself a tiny graph. If there were also a function $g : C \to B$, we could construct a diagram that looks like the above element-by-element graph, but with sets at the nodes instead of single elements:



(Fn)

So we've gone up a level: each arrow at this level could be cracked open to reveal a bundle of arrows, its own element-by-element diagram like the one we had above.

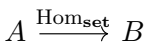
A closed function $f : A \to A$ maps from a set to itself, so we'd draw it as a loop:



(Fn)

Math in the late 1800s to early 1900s worked at this level, during which axioms were developed and used to derive a foundation for anything mathematicians could think of.

Level 3, Categories: Bundling functions But we can bundle further. In the last diagram we had a single function. But consider a collection of mappings (aka homomorphisms) from set A to set B, $\mathrm{Hom}_{\mathbf{set}}(\mathrm{A}, \mathrm{B})$. The first diagram with three elements in A and three in B could be rewired nine different ways, but it's our choice and we can include one to nine functions as desired. We would draw these bundles as



(Hom)

I think of this arrow as a bundle of quivering wires, each with its own power. If this were anime they'd be tentacles. But there is indeed a lot of

power there: with $\mathbb{R}$ representing the real numbers, we can now summarize all real one-variable functions with the diagram

$$\mathbb{R} \circlearrowright \qquad\qquad Hom \tag{Hom}$$

Level 4, Functors: Objects=categories We are approaching the level at which category theory speaks. But we can go one step further, and put a bundle of objects plus bundle of homomorphisms into a box as well, and now it is a single node in our graph. Let the above loop of all functions $f : \mathbb{R} \to \mathbb{R}$ be $\mathbf{R}$, and let's keep its company with the set of all functions on the natural numbers, $g : \mathbb{N} \to \mathbb{N}$, notated $\mathbf{N}$.

$$\mathbf{N} \xrightarrow{F} \mathbf{R} \tag{Fnctr}$$

Here, the arrow is a *functor*, which is a bundle of two sets of arrows at once: one to map elements of $\mathbb{N}$ to elements of $\mathbb{R}$, and one to map the functions $\mathbb{N} \to \mathbb{N}$ to functions $\mathbb{R} \to \mathbb{R}$. [There are additional restrictions; see below.] Custom is to write these with capital letters, to contrast with typically lower-case functions.

All these diagrams look alike, so for any diagram you come across in a textbook, it's worth checking where we are in the pyramid:

- Node is a single element; arrow represents a mapping from one element to another
- Node is a set; arrow represents a set of mappings from one set to another, aka a function
- Node is a set; arrow represents a set of functions from one set to another
- Node is a set of objects with their own sets of functions; arrow represents a functor from one context to another

2.1 Simpler categories

Some of the examples above were for rather large categories, such as all functions on all reals, but bear in mind that the same machinery works for categories of only a few objects and arrows. Say we have a square of four elements and arrows forming the square (however you want them), and a functor maps it to all reals and all functions on reals. You've selected four elements and the same number of arrows indicating the same relations. Leinster [2016] describes this as assigning data to the relation. A simple

category used for this sort of relation is formally called an *index category* or just a *diagram*.

The trivial version would be to start with the category of a single point (and its identity morphism), so mapping to a larger category is equivalent to selecting a single element.

Also, the textbooks focus on statements about morphisms that preserve all mappings, because those are the tools you want to have, but they are primarily used for situations where there are far stronger restrictions, such as those presented in Section 3, or having retracts and sections as in Section 6.

3 Bundles

We're used to thinking in terms of sets comprising atomic elements, but mathematical objects are typically a bundle of parts. Non-mathematicians think about the non-negative real numbers $\mathbb{R}^+$, but we have to start thinking about ($\mathbb{R}^+$ paired with addition) and ($\mathbb{R}^+$ paired with multiplication) as different things. Some bundles:

- *Magma*: A set S and a function $f : S \to S$.
- *Preorder*: A set S and a binary relation $\leq$. (See flash cards for the formal definition)
- *Graph*: A collection (V, A, src, tgt):
  - V: a set (vertices)
  - A: a set (arrows)
  - $src : A \to V$ a function giving arrow sources
  - $tgt : A \to V$ a function giving arrow targets

Authors who write in object-oriented programming languages will recognize this use of *object* to mean a structure of elements, some of which could be verbs.

3.1 Constraints

These objects often have constraints attached. For example, what we name the pair (S, f) depends on how restrictive we want to be about the function:

- *Magma*: A set S and a function $f : S \to S$.
- *Semigroup*: A magma where f is associative: $f(a, f(b, c)) = f(f(a, b), c)$.

- *Monoid*: A semigroup with an identity element for which $f(id, s) = s$, for all $s \in S$.
- *Group*: A monoid where every element has an inverse: $\forall x \in S$, $\exists x_{inv}$ such that $f(x, x_{inv}) = id$.

As an example, if we allow zero in $\mathbb{R}^+$, then $(\mathbb{R}^+, +)$ is a monoid (but no negative numbers, so no inverses) and $(\mathbb{R}^+, \cdot)$ is a group.

Associativity and identity are going to be part of the definition of categories, so there won't be a lot of discussion of things on the list before monoids. Groups are reserved for contexts where the sense of reversibility or symmetry make sense.

3.2 Completing a group; freedom

Say we have an element a and some function f. If $f(a, a) = a$, then we have a complete group or monoid, with a the identity and its own inverse. But say that this is not the case, and $f(a, a) = b$. Then we can incrementally fill in what is needed: the value $f(a, b) = c$, an identity where $f(a, id) = a$, $f(b, c) = d$, and so on. For example, given an element 1 and the operation $+$, completing the monoid generates the positive integers.

In these parts, the word *free* has informal, not-quite-defined usage [Leinster, 2016], but my impression is that it generally refers to the collection of the fewest parts needed to complete or generate the system. The definition of *group* above is a template; the free group is a minimal exemplar that fits that template. The word *the* is often used, because for many forms F, one free F is isomorphic to any other free F.

The *free monoid* refers to lists. Elements $\{a, b, c\}$, the empty set in the rôle of identity, and the operation of concatenation generate lists like $[a]$, $[a, b]$, $[a, b, b, a, c, c]$, ... It's a record of a walk from element to element. The free monoid with initial alphabet $\{1\}$ gets its own notation, $\underline{1}$, and is isomorphic to the positive integers.

3.3 Category constraints

A category is a bundle of elements as per Level 3 of the chain of arrow abstractions in Section 2. Bundle together some things that will be nodes (herein, objects), and the sets of mappings between some or all of them.

You get to pick which mappings you will put in the set of mappings, with a few simple conditions. You don't have to include every mapping (herein, morphism) from node to node, which as above may cover all of one-variable

high school algebra with a single arrow. But for every set we do have to include the identity morphism $id_A : A \to A$ mapping every element of an object to itself.

A category also has a composition function, so given $F : A \to B$ and $G : B \to C$, we have to include $G \circ F : A \to C$. If the composition function is the usual $g(f(a)) = c$, so this is not difficult, but in case you get creative, the textbooks also require that $id_b \circ f(a) = f(a)$ and $g \circ id(b) = g(b)$.[1]

Having these additional rules means we can't just throw out any old set of nodes and edges and call it a category, but given any old set it's not difficult to generate a complete category by adding identities and chains of functions like $h \circ g \circ f$ as needed, to the point that people usually don't even bother drawing the loops to mark the identities.

Functors, as above, are a bundle of mappings of objects to objects plus mappings from functions to functions. We want them to represent a sensible transformation from one context to another, and chaining functors is going to be central here on in, so a functor F has to have these constraints on how it maps functions and objects:

- $F(f : A \to B) = F(f) : F(A) \to F(B)$
- $F(id_A) = id_{F(A)}$
- $f \circ g = F(f) \circ F(g)$

This set of constraints is what we'll need for Level 5 below.

4 Starting to map

The nodes in all the graphs above could be anything, including the bundles of elements from Section 3. From here on, famous categories will be in boldface.

Let **PrO** be the set of all preorders and **Graph** be the set of graphs. A single preorder is a set of relations of the form $a \leq b$, and we could replace the $\leq$ symbol with an arrow, like $a \leftarrow b$, and we have another little inline graph. Doing the same for all of the elements in the preorder would generate a list of (nodes, arrows, sources, targets) in **Graph**. That is, $f : \mathbf{PrO} \to \mathbf{Graph}$ makes as much sense as $f : \mathbb{R} \to \mathbb{R}$. Set up such a mapping for every element of **PrO**, and we have a category diagram with two objects and one arrow representing all homomorphisms mapping all partial orders to graphs.

[1] Every author makes a big deal of how $G \circ F$ looks backward, but all you have to do is read $\circ$ as *of*, as in *g of f*, which is exactly how people read $g(f(x))$. Really not a big deal at all.

An arrow in a partial order maps from a single element to a single element; a monoid's function maps from two input elements to one output. One way to do this, currying, will be covered later, but our definition of category already does have one binary operation built in: the composition rule. This gives us the first example of a category that doesn't fit the intuitive approach of having a set of elements, some functions between them, and the usual function composition. Here are the components of a category built from a monoid (S, id, f):

- Objects: following the name *monoid*, this category has exactly one object. Call it §.
- Morphisms: For every monoid element M, define a morphism $M : § \to §$. Include an identity element $id : § \to §$. That is, the same mapping, $f(§) = §$, appears several times in the same category, with each arrow distinguished by different labels.
- Composition: $M_1 \circ M_2 \equiv f(M_1, M_2)$.

A category requires a composition function that is associative and correctly handles the identity morphism, and we get exactly that from the monoid requirements on f. More on this below.

Level 5: the category of categories Categories are themselves a bundle like the ones from Section 3: a set of objects paired with a set of morphisms (at this level, functors) between them. So they too could be a node in one of the above diagrams, giving us the category of categories, **Cat**.[2] Notice that the rules for functors, having an identity and associativity, is exactly the requirement we put on the set of homomorphisms inside of a single category.

Things are already getting self-referential: **PrO** $\to$ **Graph** is a node-and-arrow graph representing an operation involving the set of node-and-arrow graphs. For **Cat**, we can show that **Cat** $\to$ **Graph**, thus drawing a graph for two categories to represent how categories and graphs relate. Mathematicians love this stuff.

4.1 Types

Consider the sets $\mathbb{R}$, $\mathbb{N}$, and strings of letters A-Z, and the category of mappings between these three objects. This is the beginning of a description

[2] Authors refer to *the* category of categories, and I'm unclear whether an alternative cat-of-cats exists or what it would look like, so we may be at the top here—there's no category of categories of categories.

of a programming language. C basically runs entirely on these three types. Functions often work on structures that combine these types—at the least, arrays of several elements—which we can achieve by currying, but I'm not getting in to that. But just as a free monoid is a path along a set of elements, a program is a path selecting some functions from the category listing all options. This seems like a nice way of thinking about a lot of different types of categories.

4.2 Examples

We already mapped PrO and $Graph$, and we can go in the other direction from $Graph$ to PrO using similar thinking.

A *partial order* is a preorder that also has the condition that if $a \leq b$ then $b \not\leq a$ (when $a \neq b$). Because $\leq$ is transitive, that means we can't develop a sequence where $a \leq b$, $b \leq c$, $c \leq a$. A *tree* is a graph with no cycles, meaning exactly the same property holds for arrows in the set of trees. We therefore have the same mapping from partial orders to trees and back.

We'd like to touch every node in the tree (span the tree), by starting at a set of nodes, then following arrows to hit the next node, following all arrows from that node, and so on. Will this work starting at a single node; will it always be possible? Maybe you have an image of a tree in your head and the answer is obvious to you: no to both questions. But proving this using the four-part definition of a graph above may be a pain.

Define a minimum in a partial order to be a value such that there is no $x \neq a$ such that $x \leq a$, and to add jargon, let an ω-chain be a sequence $a \leq b \leq c \ldots$. With an infinite number of elements (like the number line), a chain may not have a minimum, and because we have a partial order, there may be multiple chains that don't intersect. Every element in a partial order is on some ω-chain, though it may be a single element by itself, or a chain with no minimum. So if there's a chain with no minimum, we can't span it. If there are only chains with minima, we can span the full set by starting at all minima, then walking up their chains.

Now use the functor mapping the category of partial orders to the category of trees, and we have set the procedure to span any given tree and given conditions for when it will work, without dealing with the graph elements V, A, src, and tgt at all. For me, I initially did it wrong with trees, because I did a proof-by-picture using a finite tree with minimum elements, but from the perspective of a partial order the full proof came more easily.

5 Where commuting gets us

It's already potentially interesting that we can express mappings between different concepts like orderings and graphs, but things get more interesting when we have multiple paths to the same place.

For example, Spivak [2014] describes multisets as sets of elements E with a set B of symbol names, and a map $\pi : E \to B$. So, for example, we could replace all the words in a book with numbers, e_1=*the*, e_2=*and*, e_{250}=*dog*, ..., and thus turn a text into a sequence like $e_1, e_{250}, e_2, e_1, \ldots$, then have a lookup function π which tells us that e_{250} maps to *dog*. We may have another set of elements and symbols in another context, maybe a corpus in another language, so we'll need an element mapping placeholders $f_1 : E \to E'$ and a symbol mapping $f_2 : B \to B'$. See the flash cards for formalization.

Here is a diagram including both the functions to map a multiset (E, B, π) to another, and the π functions inside the multisets:

$$\begin{array}{ccc} E & \xrightarrow{f_1} & E' \\ {\scriptstyle\pi}\downarrow & & \downarrow{\scriptstyle\pi'} \\ B & \xrightarrow[f_2]{} & B' \end{array} \tag{Fn}$$

The dictionary definition of *commute* is *to exchange*; in math circles, *commutativity* typically refers to reversing the order of things, like $x + y = y + x$. In this case, we can get from the original element set to the primed symbol set in two ways: either apply the symbol map π first to get to B and then step from the unprimed to primed corpus (calculating $f_2 \circ \pi(e)$), or first step to the primed corpus via f_1 and then map from E to B on that side (calculating $\pi \circ f_1(e)$).

When are both routes identical? This seems to me an interesting question in its own right. For books and word lists in Portuguese and English, I'd guess there's a natural way to map simple nouns: finding that an element in the P corpus is *cão*, then using the Portuguese-to-English dictionary to find that it means *dog* is likely equivalent to the commuted procedure of first finding the element in the Portuguese corpus matching our element of interest, then looking up that that element in the E' corpus maps to *dog*. This is going to be easy and natural.

Doing the same with *saudade* is going to take some forcing. We're going to generate a list of categories that all behave similarly and relate nicely, but what about those things that have to be thrown out to fit the categorical mold? The textbooks that focus on that mold naturally exclude them, but by definition, standardization requires jettisoning the unique.

Indeed, one way to nail down a relationship is to just cull down what is in your sets. If you need a functor mapping from preorders to partial orders, use the identity functor, and just ignore everything in the set of partial orders that isn't a preorder. Now you have a lot to say about a set of measure zero within the space of preorders.

The set of books, B, can be cataloged either via the Library of Congress classification system (a set of call numbers L), or the Dewey decimal system (another set of call numbers D). Let us pair together call numbers referring to the same book, in the cross-product space $D \times L$. The full cross product matching every Dewey number with ever LoC number makes no sense, and not all books have a classification in every system, so out of the full cross of all pairs (d, l), we will write down the relatively small list of pairs that are about the same book, herein $D \times_B L$. Finally, we have commutativity:

$$\begin{array}{ccc} D \times_B L & \longrightarrow & L \\ \downarrow & & \downarrow \\ D & \longrightarrow & B \end{array} \tag{Fn}$$

This is a *pullback*. There is information in the pairing, and maybe even information in the list of books and call numbers that can't be found from $D \times_B L$ and are now lost in the stacks.

This example may seem obvious, but in Section 5.3 we'll generalize this to other contexts.

5.1 Natural transformations

If there is a mapping from one context to another where (apply internal function, then jump) produces the same outcome as (jump, then apply internal function), we call the jump across categories a *natural transformation*. We had a set-level example above where (find English lookup, translate lookup to Portuguese) and (find Portuguese match to element in English corpus, lookup in Portuguese) had the same outcome. We can do the same at the category level.

Given two categories $\mathbf{C}$ and $\mathbf{D}$, we can define $\mathrm{Fun}(\mathbf{C}, \mathbf{D})$ as the category of functors between the two objects, and the set of arrows in that category will be the natural transformations. So defining and identifying natural transformations lets us bring the category of functors into the categorical club.

Start with two categories and two functors, F and G, each independently mapping the same first category to the same second category. For

the rest of the discussion, almost every element in the discussion to follow ($\alpha_a, \alpha_b, F(a), F(h), G(b), \ldots$ below) lives in the second category. This is different from the English/Portuguese translation, where there was no specified space in which everything lived, and f_1 and f_2 lived in an *ad hoc* space of mappings between multisets.

With that caveat and context in mind, pick two elements from the first category, a and b, and a mapping $h : a \to b$. We want mappings α_a and α_b in the second category so that

$$\begin{array}{ccc} F(a) & \xrightarrow{\alpha_a} & G(a) \\ \downarrow{\scriptstyle F(h)} & & \downarrow{\scriptstyle G(h)} \\ F(b) & \xrightarrow{\alpha_b} & G(b) \end{array} \tag{Fn}$$

The functors are a natural transformation if, for any two elements of the first category, we have αs that make this square commute. If we have all this, then we have the sort of naturalness we seek: you can start in the F-transformed version and walk from $F(a)$ to $F(b)$ and then walk over to the G-transformed version and get to $G(b)$; or you can commute the order and α-walk over to the G-transformed version first and then from $G(a)$ to $G(b)$.

Here is a failed attempt. Consider the category consisting of a single object a, one named b, and an arrow $a \to b$. We would like to map this category to ordered natural numbers, $(\mathbb{N}, \leq)$, with $\leq$ by its usual meaning. Here are two functors we can try to naturally transform:

$$\begin{aligned} F(a) &= 2 \quad ; \quad G(a) = 3 \\ F(b) &= 5 \quad ; \quad G(b) = 4 \end{aligned}$$

Both $2 \leq 5$ and $3 \leq 4$ are arrows that exist in the target category, so we have valid places to which we can map the relation $1 \leq 2$ and so we have two valid functors.

But to have a natural transformation from the F functor to the G functor, we need to find those two αs to complete the square. As noted above, they they have to be part of the set of mappings at the $(\mathbb{N}, \leq)$ category that both functors map to. There is an arrow $2 \to 3$ in that category, so we can do the walk of $F(A) \to G(A) \to G(B)$, or $2 \leq 3 \leq 4$. But we don't have $5 \leq 4$, so the walk from $F(A) \to F(B) \to G(B)$ isn't possible using the arrows the target category gave us.

Given that all our arrows go uphill, we will need transformations where $F(A)$ is downhill from both intermediate points $G(A)$ and $F(B)$, and $G(B)$

is uphill from both intermediate points. For example, given

$$\begin{aligned} F(a) = 2 \quad &; \quad G(a) = 3 \\ F(b) = 4 \quad &; \quad G(b) = 5 \end{aligned}$$

there is a natural transformation from F to G (but not from G to F).

If we wanted to start with a category with more than two elements, we would need to show that this relation holds for *any* pair of elements with an arrow between them.

5.2 Chaining

Further, all these commutative diagrams chain together. If we had English (E), Portuguese (E'), and Russian (E'') corpora, we could draw:

$$\begin{array}{ccccc} E & \xrightarrow{f_1} & E' & \xrightarrow{f_3} & E'' \\ {\scriptstyle\pi}\downarrow & & \downarrow{\scriptstyle\pi'} & & \downarrow{\scriptstyle\pi''} \\ B & \xrightarrow[f_2]{} & B' & \xrightarrow[f_4]{} & B'' \end{array} \tag{Fn}$$

It is self-evident that if the left square and the right square commute, than any flow from upper left to lower right will be equivalent. This becomes especially interesting when we have mappings across different contexts: if we can map from multisets to preorders and preorders to graphs, we just got a multiset-to-graph generator for free.

Natural transformations also chain: if there exists an n.t. from functor F to functor G, and from functor G to functor H, then we have a natural transformation from F to H. If we have three categories and natural transformations F and G from first to second, and H and I from second to third, then $H \circ F$ must have a natural transformation to $G \circ I$.

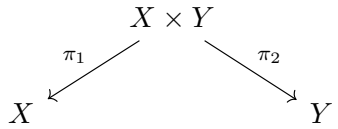
5.3 Universal properties

A universal is any statement of the form, *for all x there exists a unique y*. The ones the textbooks cover are about diagrams, as defined in Section 2.1. One diagram imposes a certain shape or set of relations onto some elements of a category. Drawing the diagram plus other objects category, you'd have the diagram as one point, and one arrow out for each constituent of the diagram, and the result looks like a cone. A limit L is the cone such that for all other cones A, there exists a unique morphism from $A \to L$. A colimit and co-cone goes in the other direction: for all A, there exists a unique

morphism $L \to A$. It is a limit because in the category of all cones, with its many morphisms from one cone to another, the limit is on the edge, with everything pointing to it.

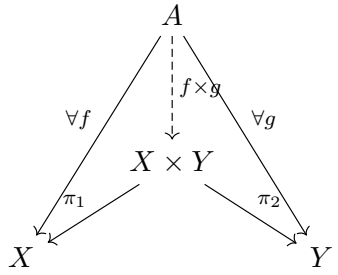
A limit is ostensibly an expression of a fundamental characteristic of a diagram in a category, summarizing information about any morphism from the diagram, and mapping that diagram to any elements of the category. On top of this, the same diagram will form a limit in multiple categories, providing another level of ostensible unification.

Consider the cross product of two sets, $A \times B$. There is an obvious mapping from $A \times B \to A$ and $A \times B \to B$—just drop the unneeded B or A part. Here is the shape of this as a diagram, with π_1 and π_2 being the *drop one part* functions:



(Hom)

This diagram (in the formal indexing set sense) is called a span and can also be turned into a little inline diagram as $X \leftarrow (X \times Y) \to Y$. Of course, there are other things we could put at the top of the fork, some other set A that maps both ways: $X \leftarrow A \to Y$. But $X \times Y$ is unique in this capacity because every A and its span can be expressed via a morphism to $X \times Y$. Here is the commutative diagram:



(Hom)

As per the annotation in the diagram, the $A \to (X \times Y)$ morphism is trivial to generate as the ordered pair $(f(a), g(a))$. For any f and g, you could go the short way to map A to X or Y, but you are guaranteed that there is also a pair of mappings $A \to (X \times Y) \to X$ and $A \to (X \times Y) \to Y$ that are the long commute to the same point. Put differently, you can have an arbitrary span composed of any object and mappings iff you can write down a map $A \to (X \times Y)$. That any span can be expressed by going through $X \times Y$ makes that object a sort of universal spanner. Returning

to the problem of combining Dewey and LoC numbers, you could come up with all sorts of other clever combination schemes besides the ordered pair (Dewey, LoC), but you are guaranteed that all of them must reduce to this simpler scheme. This sounds obvious, but people put a lot of time into trying to be clever.

You could do the same in other categories. The product of two graphs could be similarly constructed, by generating the grid of all vertex pairs $X \times Y$, then including all arrows from one pair to another if either base graph had an arrow for its corresponding vertices. Projecting back to the original graphs would also be to just drop the other dimension.

Groups seem awkward to me because they reveal how the $X \times Y \to X$ mapping isn't necessarily so neat. For $(\mathbb{R}, \cdot)$ and the span $3 \leftarrow 3 \times 4 \to 4$, the first arrow is $f_1(x) = x/4$ and the second $f_2(x) = x/3$, which is not especially general. But then, recall that even identity arrows are item specific: the arrow $3 \to 3$ and $4 \to 4$ are distinct parts of the category of $(\mathbb{R}, \cdot)$, even though we have a convenient way to express them all collectively.

Anyway, does it work? For any a, the arrow $a \to 3$ must be $f_a(a) = 3/a$, and the arrow to 3×4 must be $f_x(a) = 3 \cdot 4/a$. Commutativity works: $f_1 \circ f_x(a) = 3 \cdot 4/a \cdot 1/4 = 3/a = f_a(a)$, and the same for the other leg pointing to 4.

This gives us a nice coincidence between the set concept of $X \times Y$ and the arithmetic concept. I'd always learned it as a sort of metaphor, where if you have three dots on the x axis and twelve dots on the y then you'd have twelve (x, y) pairs. When the counts go to infinity, well, we've already got the pattern down for finite elements so don't worry about it. Here we have a more formal relationship, that any transformations into X and into Y have to have a unique corresponding transformation into $x \times y$.

For a partially ordered set, the only thing we have to work with is the $\leq$ relation. If $x \leftarrow (x \times y) \to y$ means $x \leq (x \times y) \geq y$, and we want the unique $(x \times y)$ that meets this requirement, then we need $(x \times y) \leq \min(x, y)$ to meet the requirements at all, and $\min(x, y) \leq (x \times y)$ because otherwise if $A = \min(x, y)$ then $x \leq A$ and $y \leq A$ but the unique map $A \leq (x \times y)$ won't exist.

The product notation $x \times y$ doesn't seem far-fetched for $\min(x, y)$. We could extend it to multiple elements, like $x \times y \times z = \min(x, y, z)$. We would want a multiplicative identity, which for a finite set S would be $\max(S)$, because $\min(x, \max(S)) = x$. The multiplicative identity is what you might get if you went the other way and had zero terms in your product, so the commutative diagram above reduces to *there exists a unique mapping* $A \to X \times Y$ with no additional conditions. This brings us to terminal sets.

Terminal sets A terminal set T is another universal property of a category. Here, there exists a unique mapping $A \to T$, for any A. Generally, this means that the terminal set has one element, so every element of A has to map to that single element.

Groups: for a group with only one element, that element has to be the multiplicative identity, 1.

Sets: doesn't matter the name of the element, but as above, there can be only one.

Posets: Let the single element in the terminal set be t. For any a, $a \leq t$. So, t is $\max(A)$; else if A is inifinite the terminal set doesn't exist, it seems to me.

Graphs: we need one vertex and one arrow, and the arrow must therefore have source and target equal to that one vertex. A point pointing to itself.

Back to products, all of these are indeed the multiplicative identity for the given concept of product.

We can't quite define the limit as the multiplicative identity, though that seems to be the sort of promise the authors make about the universality of universals. For example, it doesn't work for rings, for which the terminal set is $\{0\}$. There's probably a way to make this work, but it characterizes $+$, not $\times$ as the product in that context.

One way to define a thing is to just declare it: a times b is what you get when you add a items b times over. Another way is to describe the properties that the thing has. a times b is the unique thing such that the above diagram commutes, and any other transformation that hits both a and b has to go through $a \times b$. Also, it has to have a multiplicative identity, which is the thing such that $a \times id$ is isomorphic to a and there is no identity such that

6 The injective retraction epic

These are some themes that seemed to repeat at different levels. Am not sure about how to fit them in to the overall narrative.

Given a function $f : A \to B$, could we back-trace the arrows to generate a function in the opposite direction $B \to A$?

If two elements a_1 and a_2 have the same target b_1, reversing the arrows won't give us a function, because a function maps each element to only one target. If $f(a_1) = f(a_2) \Rightarrow a_1 = a_2$, the function is *injective*, and we have our condition of each reversed arrow hitting only one target. Note that we must have the size of B greater than or equal to the size of A for this

one-to-one property to hold.

Injections have a relabeling character to them, where if $f(a_1) = b_1$, then b_1 is effectively just a renaming of a_1. No information about A is lost or merged with other information, though some labels in B may not be used and their information disappears.

If there is an element of B that has no arrows pointing to it, then we can't reverse the arrows in $f : A \to B$ to get a function $B \to A$, because every element in the source set must have some target. If for all elements b, there exist some a such that $f(a) = b$, then f is surjective, aka *onto*. We must have the size of A greater than or equal to B for this to work.

Surjections have a sectioning character. Because we know that every b back-maps to some a and some bs can back-map to multiple as, we could partition the set A into elements that map to b_1, elements that map to b_2, ..., thus producing a sectioning of A. Mapping from A to B loses information about what lives in each section; all information about B is accounted for.

If f is both injective and surjective, then we can generate a proper function $f^{-1} : B \to A$. I.e., f defines an isomporphism between A and B, which must have the same number of elements. Note that in the case of infinite sets, we can effectively use this as the definition of 'same number'.

Injectivity requires that each element of B have at *most* one arrow pointing to it; surjectivity requires that each element of B have at *least* one arrow pointing to it. So given both, the back-arrows form a function where every element in B has exactly one arrow attached. We lose no information in either domain or codomain, and there is a one-to-one mapping in both directions—an isomorphism.

6.1 Retract/section

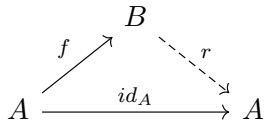
A *retraction* is a function $r : B \to A$ such that $r \circ f = 1_A$. Injectivity is necessary. Consider near-injectivity: if both $a_1 \to b$ and $a_2 \to b$ and every other $a_n \to b_n$, then we can assign $f(b_n) = a_n$ and arbitrarily assign $r(b) = r(a_1)$, but for what b is $r(b) = a_2$? We don't need surjectivity of f, because bs that don't enter into the condition that $r(f(a_n)) = a_n$ can point to wherever they please and we still have our retraction.

A *section* is a function $s : B \to A$ such that $f \circ s = 1_B$. We need surjectivity of f (the one with a sectioning character), because each element of B needs to be accounted for. We don't need injectivity, because if $a_1 \to b$ and $a_2 \to b$, arbitrarily pick $s(b) = a_1$ and the condition that $f(s(b)) = b$ works.

So we have a retraction iff f is injective; a section iff f is surjective.

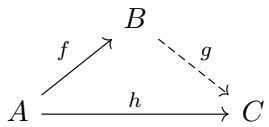
6.2 Determination and choice

Lawvere [2009] recommend drawing the retraction as



(Fn)

because now we can generalize to the determination or extension problem: does there exist a function $g : B \to C$ to fill in this diagram:

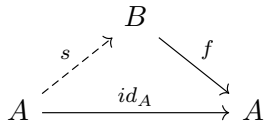


(Fn)

it's called a determination problem because $f(a)$ needs to be sufficient to derive $h(a)$—$g(\cdot)$ can't add information or recover information lost. So $h(a)$ is determined by $f(a)$—they're relabelings. If $h(a_1) \neq h(a_2)$ but $f(a_1) = f(a_2)$, $g(\cdot)$ can't help us.

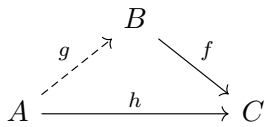
If f does have a retraction r, then $g = h \circ r$ and the triangle commutes: $g \circ f = h \circ r \circ f = h \circ 1_A$.

Similarly, draw the section as



(Fn)

because now we can generalize to the choice or lifting problem: does there exist a function $f : A \to B$ to fill in this diagram:



(Fn)

The same sectioning story holds in the general case. There could be multiple elements of B each pointing to $h(a)$, and $g(a)$ has to choose one of those.

If f has a section s, then $g = s \circ h$ and the triangle commutes: $f \circ g = f \circ s \circ h = 1_B \circ h$.

6.3 Epimorphisms and monomorphisms, limits

An epimorphism is a function f such that, given this diagram,

$$X \xrightarrow{f} Y \underset{g_2}{\overset{g_1}{\rightrightarrows}} Z \qquad \text{(Hom)}$$

$f \circ g_1 = f \circ g_2 \Rightarrow g_1 = g_2$. If f is not surjective, then this breaks: non-surjectivity means we have some y such that there is no $f(x) = y$, and we can construct $g_1(y) = z_1$ and $g_2(y) = z_2$. So surjectivity is necessary, and it seems obvious that surjectivity is sufficient. We could have many sections in X, but once we've reduced it to one representative in Y per section, we won't need to recover the details of each section.

F is *monic* or *mono* iff $f \circ g_1 = f \circ g_2 \Rightarrow g_1 = g_2$ in:

$$Z \underset{g_2}{\overset{g_1}{\rightrightarrows}} Y \xrightarrow{f} X \qquad \text{(Hom)}$$

Now a failure of injectivity kills the requirement, because if $f(y_1) = f(y_2)$ for $y_1 \neq y_2$, it is easy to construct $g_1(z_1) = y_1$ and $g_2(z_1) = y_2$ and the monic condition fails. Injectivity is sufficient because f injective means the reachable elements of Z are just relabelings of Y.

Epis are limits monos are colimits, as per the discussion of limits above. That fits in with the information loss stories above: if you have some scheme that might lose more or less information and will still allow the equalization properties above, it'll be equivalent to going through the epis and monos, which therefore indicates that those other schemes lose as much information.

6.4 Summary

Injective:

- Every element of the codomain has at most one back-arrow to the domain.
- Those elements in the codomain that are pointed to are effectively re-labels of the points in the domain.
- The domain must have as many or more elements than the codomain.
- All information in domain is retained; information could be lost in codomain.
- We can write a retraction such that we can get back to the original elements of A, meaning $r \circ f = 1_A$, by just following the arrows back.
- Gives us the monomorphism because an initial relabeling won't affect whether g_1 and g_2 match or don't.

Surjective:

- Every element of the codomain has at least one back-arrow to the domain.
- The domain may have multiple arrows pointing to the same target, thus splitting the domain into sections based on target elements.
- The codomain must have as many or more elements than the domain.
- All information in codomain is retained; information may be lost from the domain.
- We can write a section such that $f \circ s = 1_B$, because surjectivity gives us that each element has an arrow, and if $f(a_1) = f(a_2) = b$, just pick one.
- The epimorphism condition is met because every element of Y is relevant to the monomorphism chain.

Injective + Surjective = Isomorphism:

- Every element of codomain has exactly one back-arrow
- Codomain elements are unique relabels of domain elements; all codomain labels are used.
- Domain and codomain are the same size (for most authors, by definition).
- No information in domain or codomain is lost.
- We can write f^{-1} such that $f^{-1} \circ f = 1_A$ and $f \circ f^{-1} = 1_B$.
- The function f is both epic and monic, as no information is lost on either end.

References

Steve Awodey. *Category theory*. Oxford University Press, Oxford New York, 2010.

F. W. Lawvere. *Conceptual Mathematics: A First Introduction to Categories*. Cambridge University Press, Cambridge, UK New York, 2009.

Tom Leinster. Basic category theory. 2016.

Emily Riehl. *Category theory in context*. Dover Publications, Inc, Mineola, New York, 2016.

David Spivak. *Category Theory for the Sciences*. The MIT Press, Cambridge, Massachusetts, 2014.