FIBRATIONS, CLEAVAGES, AND LENSES

BARTOSZ MILEWSKI

In category theory, as in life, you spend half of your time trying to forget things, and half of the time trying to recover them. A morphism, the basic building block of every category, is like a defective isomorphism. It maps the initial state to the final state, but it provides no guarantees that you can recover the original. But it seems like this lossiness is what makes morphisms useful.

There are people who can memorize mathematical formulas perfectly but have no idea what they mean. And there are those who get just the gist of it, but can derive the rest when needed. Somehow *understanding* is related to lossy compression.

We can't recover lost information. Once it's gone, it's gone. All we can do is to try to figure out what the original might have been like. In fact, knowing how the information was lost, we might be able to generate all possible inputs that could have led to a given output. It's the closest we can get to inverting the uninvertible. This is the main idea behind fibrations.

Let me illustrate this concept with an example. Consider the function `isEven`:

```
isEven :: Integer -> Bool
isEven n = n `mod` 2 == 0
```

This function is definitely not invertible. If I only told you that the output was `True`, you couldn't tell me what the input was. You could, however, give me the set of all inputs that could have produced this output: it's the set of even numbers. We often call this set, which is the inverse image of `True`, a *fiber* over `True`. Similarly, the fiber over `False` is the set of odd numbers. In this case we only have only two fibers and they happen to be isomorphic.

Here's a more interesting example. Consider a set of all lists of integers and a function that returns the length of a list: a natural number:

```
length :: [Integer] -> Nat
```

This function is not invertible, but it defines fibers over natural numbers. The fiber over zero is a one-element set that contains only the empty list. The fiber over 1 is the set of lists of length one (which is isomorphic to the set of integers). The fiber over 2 is a set of 2-element lists, or pairs of integers, and so on. You may recognize these fibers as length-indexed lists, or vectors. You can find them, for instance, in the Haskell Vec library or as `Vect` in Idris. These are not your typical data types, though. They are examples of *dependent types*–types that depend on values (here, natural numbers).

Notice that the name "length-indexed lists" suggests a slightly different interpretation of these types. You may think of them as families of types parameterized by natural numbers. This would suggest a mapping from elements of a set (natural numbers) to types. These two views are equivalent, but in category theory we try to avoid, if possible, talking about sets (and set elements in particular). The interpretation of dependent types as fibrations is more general, so let's dig into fibrations.

As a generalization of functions like `isEven` or `length`, we'll consider a morphism $\pi\colon e \to b$, and call it a projection, since it projects each fiber down to one element. Our goal, though, is to define a fiber as the pre-image of an element in b. But what's an

element? The closest we can get to defining an element in category theory is to consider a morphism x from the terminal object 1. Such morphism is called a *global element* and, in Set it really picks a single element from a (non-empty) set. Now we have two morphisms converging on b: π and x. Conceptually, a fiber is a subobject e_x of e, which means that there must be a morphism s that embeds e_x in e. Moreover, when this embedding is followed by the projection π, it must produce the same element as x. The best such object is given by a universal construction which, in this case, is a pullback.

$$\begin{array}{ccc} e_x & \xrightarrow{s} & e \\ {\scriptstyle !}\downarrow & \lrcorner & \downarrow{\scriptstyle \pi} \\ 1 & \xrightarrow{x} & b \end{array}$$

(The exclamation mark stands for the unique morphism to the terminal object.) Incidentally, this is why a pullback is sometimes called a *fiber product*.

If fibers over all elements x are isomorphic, the pair (e, π) is, quite fittingly, called a *fiber bundle*. The object b, from which the fibers sprout, is called the base. (Notice that lenght-indexed lists don't form a bundle.)

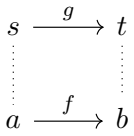
Anything you can do with functions, you can do with functors, only better. So we can have a category E, another category B, and a functor $p\colon E \to B$. Since a functor acts as a function on objects (modulo size issues), we can define a fiber as a set of objects in E that are mapped to a single object in B. The big question is, what do we do with morphisms? We have potentially lots of morphisms in E that go between any two fibers, and which get projected down to a single hom-set in B. If we want to invert p, we have to design a procedure for lifting morphisms from B to E.

Here's the idea: We would like each fiber to form a subcategory of E, and we'd like to pick morphisms between fibers in such a way that p^{-1} becomes a functor from B to Cat. In other words, we want p^{-1} to map objects of B to categories (the fibers), and morphisms of B to functors between those categories. If this is too much to ask (which it often is), we'll settle for p^{-1} to be a *pseudo-functor*, which is a functor that preserves unit and composition only up to isomorphism. In fact the original construction (attributed to Grothendieck) produces a *contravariant* pseudo-functor. In this post I'll describe the *covariant* version of this construction, which is called *opfibration*, and which is easier to explain.

The starting point of Grothendieck fibration is the recipe for lifting morphisms from the base category B to the total category E. There is a universal construction for doing that. The resulting morphisms are called *opcartesian*.

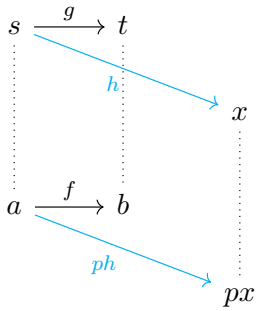
Let's start with a morphism $f\colon a \to b$ in the base category and pick an arbitrary object s (source) in the fiber over a (hence $a = ps$). This will be the source of our opcartesian morphism. We have a lot of choices for the target. Strictly speaking, the target should be one of the objects *over* b, and that's what we are aiming for. However, a universal construction should look at a much larger pool of candidates, some of them with targets in other fibers. This pool enlargement helps narrow down the final choice with greater accuracy. (Remember, universal constructions are unique only up to isomorphism.)

The opcartesian morphism over f, with the specified source s, is a morphism $g\colon s \to t$, such that $pg = f$.

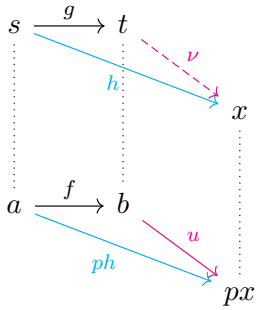


It must satisfy a universal property that I'm about to describe.

First, we pick an arbitrary object x and a morphism $h\colon s \to x$. This is supposed to be the competition for g. When projected down to B, it becomes $ph\colon a \to px$.



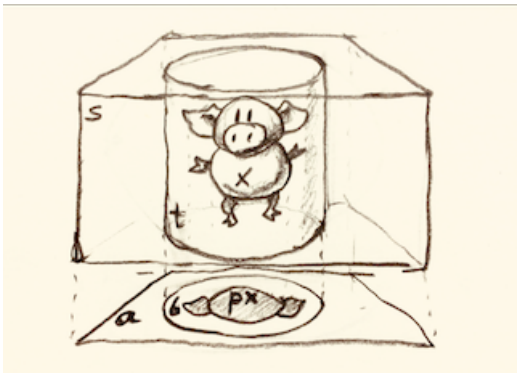
We are interested in the case when ph factorizes through f, that is, there is a morphism $u\colon b \to px$ such that $ph = u \circ f$. Whenever such factorization is possible in B, we demand that there be a unique lifting of it to E.



In other words, there exists a unique $\nu\colon t \to x$ such that $h = \nu \circ g$ and $p\nu = u$.

If you find this definition a little confusing, you're in good company. So let's try a slightly different imagery that has more to do with the original ideas from algebraic topology. Think of objects as shapes. A morphism $f\colon a \to b$ is a proof that b is a proper subset of a, or that a *contains* b. A functor between two categories of shapes must map shapes to shapes in a way that preserves inclusion. It may map many shapes to one, so imagine that the shapes in the category E are three-dimensional, and their projections using the functor p are their flat shadows. Functoriality means that, if s contains t, then its shadow $a = ps$ contains $b = pt$.

Next, we introduce a new object x that is contained inside s, and the proof of that is $h\colon s \to x$. It follows from functoriality that a contains the shadow of x.



Now suppose that this shadow falls inside the smaller b (with the proof $u\colon b \to px$). Normally, this would not imply that x is inside t. It's possible that (parts of) x are sticking out below or above t. Our universal condition demands that this cannot happen. There can be no room above or below t– it's a cylinder carved into s. Universality guarantees that we get the absolutely optimal shape.

Now that we know what an opcartesian morphism is, we might ask the question, does it always exist? Given an arbitrary morphism $f\colon a \to b$ in B and an object s over a, can we always find an opcartesian morphism $g\colon s \to t$ such that t is over b? If we can, then we call the pair $(E, p\colon E \to B)$ an *opfibration*.

Here's an interesting observation. You might wonder whether the definition of an opcartesian morphism isn't overly complicated. Wouldn't it be enough to restrict the pool of possible candidates to those morphisms whose target, the x in our picture, was over b, the target of f? This was, in fact, the original idea in the Grothendieck construction. The problem was that, with such definition, there was no guarantee that a composition of two opcartesian morphisms would be again opcartesian. The current definition makes that automatic.

Given an opfibration, we now face the opposite problem: there may be too many opcartesian morphisms. Remember, we wanted to (a) make fibers into subcategories of E and (b) use opcartesian morphisms to define functors between them. The first part is relatively easy: a fiber E_a has, as objects, those objects of E whose projection is a. We select as morphisms in E_a those morphisms that project down to identity, id_a (notice that we ignore other endomorphism $a \to a$). These are called *vertical* morphisms. But to define functors between fibers we need to map each object of one fiber to exactly one object in the other fiber (and the same for vertical morphisms). Think of this as *transporting* objects between fibers. In a fibered category, we could use opcartesian morphisms for transport. Any time two objects are connected in the base by a morphism, we have a bunch of opcartesian morphisms over it starting from every single object in the source fiber. We could use them to try to define a functor between fibers.

But in general we have more than one opcartesian morphism between a source object in one fiber and candidate target objects in the other fiber. But we can just pick one (if you're into set theory, you'll notice that we have to use the Axiom of Choice). Such choice is called an *opcleavage*, and the resulting construction is called *cloven opfibration*.

Formally, an opcleavage is described by a function $\kappa(f, s)$

$$\begin{array}{ccc} s & \xrightarrow{\kappa(f,s)} & t \\ \vdots & & \vdots \\ a & \xrightarrow{f} & b \end{array}$$

It takes a morphism $f\colon a \to b$ and an object s (such that $ps = a$), and produces an object t (such that $pt = b$), which is the target of some opcartesian morphism $s \to t$. This is exactly the morphism selected by opcleavage.

The universal construction of opcartesian morphisms can then be used to define the mapping of vertical morphisms thus completing the definition of a functor between fibers.

A geometric intuition is that an opcleavage provides a way of transporting objects in the horizontal direction. Vertical morphisms transport objects vertically, and the functors defined by the opcleavage transport them horizontally, in such a way that their shadows follow the arrows in the base. The origin of this intuition goes back to differential geometry, where one is able to define *continuous* paths in the base manifold and use them to transport objects, such as vectors, between fibers. Category theory lets us abstract away continuity (and differentiability) from this picture. You might also see transport used in homotopy type theory, with paths standing for equality proofs.

Now, remember what I said about the composition of opcartesian morphisms resulting in an opcartesian morphism? Unfortunately, once we start picking individual morphisms to construct an opcleavage, this compositionality might be lost. The composition of any two morphisms from the selected pool is still opcartesian, but it's not necessarily part of the opcleveage. This is why we might have to relax compositionality and embrace pseudofunctors.

But sometimes an opcleavage preserves compositionality. We call this situation *split opfibration*. It must satisfy these two conditions:

$$\kappa(id_a, s) = id_s$$

$$\kappa(f', s') \circ \kappa(f, s) = \kappa(f' \circ f, s)$$

for any $f\colon a \to b$ and $f'\colon b \to c$.

A split opfibration defines a functor $B \to Cat$, which maps objects from the base category to fibers seen as categories; and morphisms from the base category to functors between those fibers. So defined functor may be interpreted as an attempt at inverting the original projection $p\colon E \to B$.

If the splitting conditions are satisfied only up to isomorphism, we get a pseudofunctor $B \to Cat$. This makes things more complicated but also more interesting. It means that horizontal transport depends on the path. In particular, transport along a closed path–a chain of morphisms in the base that compose to identity–may produce an object that's different from (albeit isomorphic to) the starting object. In differential geometry we would say that the space has non-zero curvature.

Interestingly, this procedure of generating split opfibrations has its inverse. Given a functor $B \to Cat$ it's possible to reconstruct the total category E and a projection $p\colon E \to B$. This is called the Grothendieck construction.

Since our new slogan is "lenses are everywhere," it should come as no big surprise that a split opfibration may be seen as a type of a lens. The projection corresponds to *view* or *get*. It extracts a, the focus of the lens, out of s. The opcleavage part of opfibration, $\kappa(f, s)$ corresponds to *put* or, more precisely to *over*. It takes a morphism that modifies the focus from a to b and it takes the object s, and produces the new object t. In programming, *get* and *put* are just functions between sets, here they are object mappings of two functors, but the similarity is hard to ignore.

1. Acknowledgment

I'm grateful to Bryce Clarke for reading the draft and helpful comments.

2. Papers to read

- Johnson, Rosebrugh, and Wood, Lenses, fibrations and universal translations
- Johnson and Rosebrugh, Delta lenses and opfibrations