

29

Topoi

I REALIZE THAT WE MIGHT be getting away from programming and diving into hard-core math. But you never know what the next big revolution in programming might bring and what kind of math might be necessary to understand it. There are some very interesting ideas going around, like functional reactive programming with its continuous time, the extension of Haskell's type system with dependent types, or the exploration on homotopy type theory in programming.

So far I've been casually identifying types with *sets* of values. This is not strictly correct, because such an approach doesn't take into account the fact that, in programming, we *compute* values, and the computation is a process that takes time and, in extreme cases, might not terminate. Divergent computations are part of every Turing-complete language.

There are also foundational reasons why set theory might not be the best fit as the basis for computer science or even math itself. A good analogy is that of set theory being the assembly language that is tied

to a particular architecture. If you want to run your math on different architectures, you have to use more general tools.

One possibility is to use spaces in place of sets. Spaces come with more structure, and may be defined without recourse to sets. One thing usually associated with spaces is topology, which is necessary to define things like continuity. And the conventional approach to topology is, you guessed it, through set theory. In particular, the notion of a subset is central to topology. Not surprisingly, category theorists generalized this idea to categories other than **Set**. The type of category that has just the right properties to serve as a replacement for set theory is called a *topos* (plural: *topoi*), and it provides, among other things, a generalized notion of a subset.

29.1 Subobject Classifier

Let's start by trying to express the idea of a subset using functions rather than elements. Any function f from some set a to b defines a subset of b —that of the image of a under f . But there are many functions that define the same subset. We need to be more specific. To begin with, we might focus on functions that are injective — ones that don't smush multiple elements into one. Injective functions “inject” one set into another. For finite sets, you may visualize injective functions as parallel arrows connecting elements of one set to elements of another. Of course, the first set cannot be larger than the second set, or the arrows would necessarily converge. There is still some ambiguity left: there may be another set a' and another injective function f' from that set to b that picks the same subset. But you can easily convince yourself that such a set would have to be isomorphic to a . We can use this fact to define a subset as a family of injective functions that are related by isomorphisms of their

domains. More precisely, we say that two injective functions:

$$\begin{aligned} f &:: a \rightarrow b \\ f' &:: a' \rightarrow b \end{aligned}$$

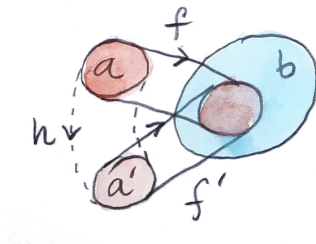
are equivalent if there is an isomorphism:

$$h :: a \rightarrow a'$$

such that:

$$f = f' \cdot h$$

Such a family of equivalent injections defines a subset of b .



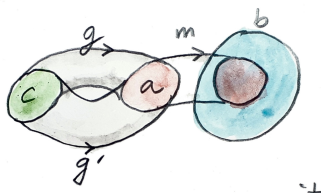
This definition can be lifted to an arbitrary category if we replace injective functions with monomorphism. Just to remind you, a monomorphism m from a to b is defined by its universal property. For any object c and any pair of morphisms:

$$\begin{aligned} g &:: c \rightarrow a \\ g' &:: c \rightarrow a \end{aligned}$$

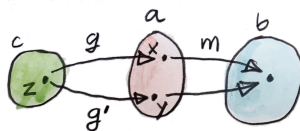
such that:

$$m \cdot g = m \cdot g'$$

it must be that $g = g'$.



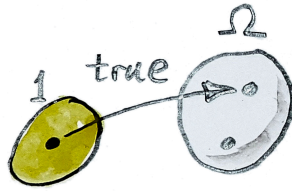
On sets, this definition is easier to understand if we consider what it would mean for a function m *not* to be a monomorphism. It would map two different elements of a to a single element of b . We could then find two functions g and g' that differ only at those two elements. The post-composition with m would then mask this difference.



There is another way of defining a subset: using a single function called the characteristic function. It's a function χ from the set b to a two-element set Ω . One element of this set is designated as “true” and the other as “false.” This function assigns “true” to those elements of b that are members of the subset, and “false” to those that aren't.

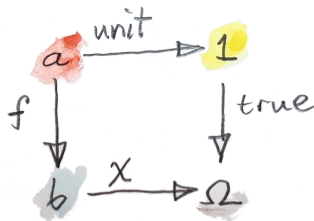
It remains to specify what it means to designate an element of Ω as “true.” We can use the standard trick: use a function from a singleton set to Ω . We'll call this function *true*:

$$true :: 1 \rightarrow \Omega$$



These definitions can be combined in such a way that they not only define what a subobject is, but also define the special object Ω without talking about elements. The idea is that we want the morphism *true* to represent a “generic” subobject. In **Set**, it picks a single-element subset from a two-element set Ω . This is as generic as it gets. It’s clearly a proper subset, because Ω has one more element that’s *not* in that subset.

In a more general setting, we define *true* to be a monomorphism from the terminal object to the *classifying object* Ω . But we have to define the classifying object. We need a universal property that links this object to the characteristic function. It turns out that, in **Set**, the pullback of *true* along the characteristic function χ defines both the subset *a* and the injective function that embeds it in *b*. Here’s the pullback diagram:



Let’s analyze this diagram. The pullback equation is:

$$true \cdot unit = \chi \cdot f$$

The function $\text{true} \cdot \text{unit}$ maps every element of a to “true.” Therefore f must map all elements of a to those elements of b for which χ is “true.” These are, by definition, the elements of the subset that is specified by the characteristic function χ . So the image of f is indeed the subset in question. The universality of the pullback guarantees that f is injective.

This pullback diagram can be used to define the classifying object in categories other than **Set**. Such a category must have a terminal object, which will let us define the monomorphism true . It must also have pullbacks — the actual requirement is that it must have all finite limits (a pullback is an example of a finite limit). Under those assumptions, we define the classifying object Ω by the property that, for every monomorphism f there is a unique morphism χ that completes the pullback diagram.

Let’s analyze the last statement. When we construct a pullback, we are given three objects Ω , b and 1 ; and two morphisms, true and χ . The existence of a pullback means that we can find the best such object a , equipped with two morphisms f and unit (the latter is uniquely determined by the definition of the terminal object), that make the diagram commute.

Here we are solving a different system of equations. We are solving for Ω and true while varying both a and b . For a given a and b there may or may not be a monomorphism $f :: a \rightarrow b$. But if there is one, we want it to be a pullback of some χ . Moreover, we want this χ to be uniquely determined by f .

We can’t say that there is a one-to-one correspondence between monomorphisms f and characteristic functions χ , because a pullback is only unique up to isomorphism. But remember our earlier definition of a subset as a family of equivalent injections. We can generalize it by defining a subobject of b as a family of equivalent monomorphisms to

b . This family of monomorphisms is in one-to-one correspondence with the family of equivalent pullbacks of our diagram.

We can thus define a set of subobjects of b , $Sub(b)$, as a family of monomorphisms, and see that it is isomorphic to the set of morphisms from b to Ω :

$$Sub(b) \cong C(b, \Omega)$$

This happens to be a natural isomorphism of two functors. In other words, $Sub(-)$ is a representable (contravariant) functor whose representation is the object Ω .

29.2 Topos

A topos is a category that:

1. Is Cartesian closed: It has all products, the terminal object, and exponentials (defined as right adjoints to products),
2. Has limits for all finite diagrams,
3. Has a subobject classifier Ω .

This set of properties makes a topos a shoe-in for **Set** in most applications. It also has additional properties that follow from its definition. For instance, a topos has all finite colimits, including the initial object.

It would be tempting to define the subobject classifier as a coproduct (sum) of two copies of the terminal object –that’s what it is in **Set**– but we want to be more general than that. Topoi in which this is true are called Boolean.

29.3 Topoi and Logic

In set theory, a characteristic function may be interpreted as defining a property of the elements of a set — a *predicate* that is true for some elements and false for others. The predicate *isEven* selects a subset of even numbers from the set of natural numbers. In a topos, we can generalize the idea of a predicate to be a morphism from object a to Ω . This is why Ω is sometimes called the truth object.

Predicates are the building blocks of logic. A topos contains all the necessary instrumentation to study logic. It has products that correspond to logical conjunctions (logical *and*), coproducts for disjunctions (logical *or*), and exponentials for implications. All standard axioms of logic hold in a topos except for the law of excluded middle (or, equivalently, double negation elimination). That's why the logic of a topos corresponds to constructive or intuitionistic logic.

Intuitionistic logic has been steadily gaining ground, finding unexpected support from computer science. The classical notion of excluded middle is based on the belief that there is absolute truth: Any statement is either true or false or, as Ancient Romans would say, *tertium non datur* (there is no third option). But the only way we can know whether something is true or false is if we can prove or disprove it. A proof is a process, a computation — and we know that computations take time and resources. In some cases, they may never terminate. It doesn't make sense to claim that a statement is true if we cannot prove it in finite amount of time. A topos with its more nuanced truth object provides a more general framework for modeling interesting logics.

29.4 Challenges

1. Show that the function f that is the pullback of *true* along the characteristic function must be injective.