[**Relations are hard to model in category theory**](https://0fps.net/2013/06/04/relations-are-hard-to-model-in-category-theory/)

Posted on [June 4, 2013](https://0fps.net/2013/06/04/relations-are-hard-to-model-in-category-theory/)by [mikolalysenko](https://0fps.net/author/mikolalysenko/)

WARNING: This is a somewhat rambling post about category theory.  If half-baked mathematical philosophy is not your thing, please consider navigating away right now.

Anyway, the thing that I want to write about today is the difference between category theory and what I shall call for lack of a better term the older “set theoretic” approach.  My goal is to try to articulate what I see as the main difference is between these two structures, and why I think that while category theory offers many insights and new perspectives it is probably hopeless to try to shoe horn all of mathematics into that symbolism.

Relation Theory

If you’ve ever taken an advanced math course, you probably already know that [set theory](http://en.wikipedia.org/wiki/Set_theory) is default “programming language” of modern mathematics.  Modern set theory is built upon two basic components:

* [Sets](http://en.wikipedia.org/wiki/Set_(mathematics))
* [Relations](http://en.wikipedia.org/wiki/Relation_(logic))

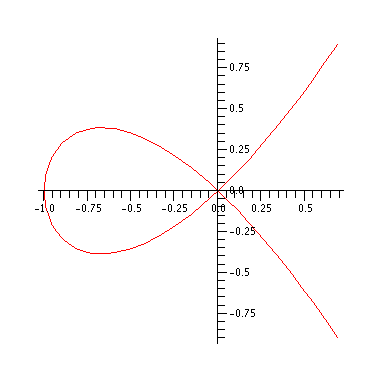
A set is any well defined unordered collection of objects, the members of which are called its elements.  Sets by themselves are rather boring things, and don’t do much on their own.  What makes set theory interesting and useful, is that in addition to sets we also have relations.  There are many ways to define a relation, but if you already know set theory we can bootstrap the definition using the concept of a [Cartesian product](http://en.wikipedia.org/wiki/Cartesian_product):

An n-ary relation amongst the sets X_1, X_2, ..., X_n is a subset of R \subseteq X_1 \times X_2 \times ... \times X_n.  A tuple of n elements, x_1 \in X_1, x_2 \in X_2, ..., x_n \in X_n is related (under *R*) if and only if,

R(x_1, x_2, ..., x_n) \Leftrightarrow (x_1, x_2, ... ) \in R

This may seem a bit circular, but it is unavoidable since we need some formal language to define set theory, and without set theory don’t really have any way to talk about formal languages!  The only way out is to take at least something as a given, and for most mathematicians this is the definition of sets and relations.

The main goal of set theory is to define new sets and the relations between them.  A classic example of a relation is a graph, which in the plane visualizes relationships between a pair of variables:

[](https://0fps.files.wordpress.com/2013/06/nodal_cubic.gif)

A nodal cubic curve graphed as a relation between two variables x and y

Relations show up all over the place in mathematics.  For example, we have binary relations like =, <, >, etc. that we can use to compare quantities.  It is also possible to think of arithmetic in terms of relations, for example + can be thought of as a ternary relation that takes 3 numbers as input and checks if the 3rd is the sum of the two inputs.

It is possible to build up more complex relations in set theory by combining simple relations using the quantifiers [there-exists](https://en.wikipedia.org/wiki/Existential_quantification) and [for-all](http://en.wikipedia.org/wiki/Universal_quantification). For example using the multiplication relation we can write the statement “2 divides x” using a quantifier:

\exists y \in \mathbb{N}:\times(2,y,x)

Where I am using  somewhat non-standard notation here to write multiplication as a ternary relation:

\times(a, b, c) \cong (a b = c)

Relational thinking is also the main concept behind [tensor calculus](http://en.wikipedia.org/wiki/Tensor), where one replaces all the sets involved with [vector spaces](http://en.wikipedia.org/wiki/Vector_space) and the relations with [multilinear forms](http://en.wikipedia.org/wiki/Multilinear_form).  In fact, the whole concept of the [Galerkin method](http://en.wikipedia.org/wiki/Galerkin_method) in numerical analysis can be thought of as first relaxing a set theoretic problem into a tensor relation; then performing a finite expansion in some basis.

Category Theory

You can approach category theory in many ways, but coming from set theory the easiest way to understand it is that it is the study of [functional relations](http://en.wikipedia.org/wiki/Function_(mathematics)) above all else.  The basic axioms of a [category](http://en.wikipedia.org/wiki/Category_theory) define a mathematical structure in which we can study certain abstract classes of functions.  Briefly a category has three pieces of data:

* A set of objects, \text{Ob}(C)
* For every pair of objects a, b \in \text{Ob}(C) set of morphisms, \text{Mor}_C(a,b)
* A relation \circ \subseteq \text{Mor}_C(a,b) \times \text{Mor}_C(b,c) \times \text{Mor}_C(a,c) for every triple of objects a,b,c \in \text{Ob}(C)

Such that the following conditions are satisfied:

* \circ is a functional relation, \text{Mor}_C(a,b) \times \text{Mor}_C(b,c) \to \text{Mor}_C(a,c)
* There exists some 1_a \in \text{Mor}_C(a,a) such that \forall f\in\text{Mor}_C(a,b):f\circ 1_a=f
* \circ is associative, that is f \circ (g \circ h) = (f \circ g) \circ h

Categories play a central role in algebra, where they are used to express transformations between various structures.  Perhaps the place where this is most obviously useful is in the study of [groups](http://en.wikipedia.org/wiki/Group_theory) and their [representations](http://en.wikipedia.org/wiki/Representation_theory).  Also the fact that many common algebraic structures like [monoids](http://en.wikipedia.org/wiki/Monoid) are secretly just degenerate versions of categories, highlights their central importance.  Unlike relations categories have a great deal of structure, which makes it possible to say much more about them than one can about a relation in general.  It can be difficult to cast a relational problem into the language of categories, but the payoff is generally worth it.  For example, one of the more successful ways to study tensor algebra today is from the perspective of [algebraic geometry](http://en.wikipedia.org/wiki/Algebraic_geometry).

Categories vs Relations

The main difference between categories and relations is that categories focus on **change**, while relations express **invariance**.  Both philosophies are equal in their overall expressive power, but they may be better suited to some problems over others.

The main power of category theory is that it lends itself to explicit calculations and so it is most useful as a language for describing transformations.  This makes it an especially nice language for reasoning about algorithms and programs, and one can see this in languages like [Haskell](http://www.haskell.org/haskellwiki/Haskell).  On the other hand, relations make minimal assertions about how we might compute something and instead only describe “what” is to be computed.  Languages like [SQL](http://en.wikipedia.org/wiki/SQL) or [Prolog](http://en.wikipedia.org/wiki/Prolog) make heavy use of relations and are good at expressing data processing concepts.

For example, it is trivial to convert any problem in category theory into the language of relations (this is vacuously easy, since the axioms of a category are specified in terms of sets and relations!)  However, going the other way is a more complicated proposition and there is no perfect solution.  Perhaps the most direct way is to “emulate” relational reasoning within category theory, which can be done using the language of [monoidal categories](http://en.wikipedia.org/wiki/Monoidal_category).  However, simply shoe horning relational thinking into this framework loses most of the advantage of categorical reasoning.  It is similar to the argument that you can take any program written in [BrainFuck](https://en.wikipedia.org/wiki/Brainfuck) and port it to Haskell by simply writing an interpreter in Haskell.  While it would then be strictly true that doing this translates BrainFuck to Haskell, it misses the point since you are still basically coding in BrainFuck (just with an extra layer of abstraction tacked on).

Categorification is hard

This is really why [categorification](http://en.wikipedia.org/wiki/Categorification) (and programming in general) is hard: there is always more than one way to do it, and the more general you get the worse the result.  Effectively categorifiying various structures requires thinking deeply about the specific details fo the situation and picking axioms which emphasize the important features of a problem while neglecting the superfluous details.

Conclusion

In the end, one view point isn’t any better than the other – but they are different and it is worth trying to understand both deeply to appreciate their strengths and weaknesses.  Relational thinking is successful when one needs to reason about very complicated structures, and is pervasive in analysis.  Categories on the other hand bring with them much more structure and precision, and are most useful in describing transformations and syntactic abstraction.