Turing Categories & Computability

An abstract look at computability

Amittai Siavava

May 24, 2024

Dartmouth Math 29

Table of contents

Categories

What is Category Theory Anyway?

You may have noticed that different branches of mathematics seem to share similar structures and, sometimes, even results.

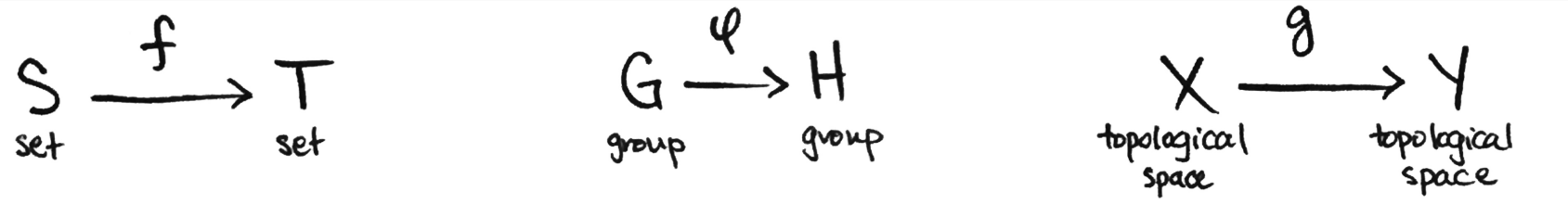

Figure 1: sets, groups, spaces...

Can these similarities and differences teach us anything...
...at all...
about these structures?

What is Category Theory Anyway?

Definition

A **category** $\mathcal{C}$ consists of:

1. A collection $\mathbf{ob}(\mathcal{C})$ of **objects**;
2. For each pair of objects $A, B \in \mathbf{ob}(\mathcal{C})$, a set $\mathcal{C}(A, B)$ of **morphisms** (aka "**arrows**") from A to B;
3. For each object $A \in \mathbf{ob}(\mathcal{C})$, a morphism $\mathrm{id}_A \in \mathcal{C}(A, A)$;
4. For each triple of objects $A, B, C \in \mathbf{ob}(\mathcal{C})$, a composition function

$$\circ : \mathcal{C}(B, C) \times \mathcal{C}(A, B) \to \mathcal{C}(A, C).$$

...preserving associativity: $f \circ (g \circ h) = (f \circ g) \circ h$

...and identity: $\mathrm{id}_B \circ f = f = f \circ \mathrm{id}_A$.

Categories... Examples?

Category's name:	its objects:	its morphisms:
Set	sets	functions
Group	groups	group homomorphisms
Top	topological spaces	continuous functions
Vect_k	vector spaces over a field, k	linear transformations
Meas	measurable spaces	measurable functions
Poset	partially ordered sets	order-preserving functions
Man	smooth manifolds	smooth maps

Category Theory is all about Relationships!

Category theory strips away *a lot* of detail.
By ignoring the details, attention is diverted to **relationships** between objects and morphisms.

Definition

A **functor** $F : \mathcal{C} \to \mathcal{D}$ between categories $\mathcal{C}$ and $\mathcal{D}$ consists of:

1. A function $F : \mathbf{ob}(\mathcal{C}) \to \mathbf{ob}(\mathcal{D})$;
2. For each pair of objects $A, B \in \mathbf{ob}(\mathcal{C})$, a function

$$F : \mathcal{C}(A, B) \to \mathcal{D}(F(A), F(B)).$$

...preserving composition and identity.

Functors... Examples?

A functor called π_1 associates a group to every topological space:

$$\begin{array}{ccc} \text{Top} & \longrightarrow & \text{Group} \\ X & \longmapsto & \pi_1(X) \end{array}$$

"the fundamental group of X"

You can use this functor to prove

1. the fundamental theorem of algebra — every nonconstant polynomial in $\mathbb{C}[z]$ has a root in $\mathbb{C}$.
2. Brouwer's Fixed Point Theorem — every homeomorphism of the disc has a fixed point
3. the Perron–Frobenius theorem — every 3x3 matrix with positive real entries has a positive eigenvalue

to name a few!

Turing Categories

Turing Categories?

Are you saying that...

If you put codes for Turing machines in a...

thing...

you get a category?

We need to address a few things.

1. morphisms? or objects?
2. partiality?
3. what about the system of coding?

Yes, Turing Categories!

But we need some more structure

Specifically, we need:

1. A **restriction structure** to handle partiality;
   Assigns to each morphism $f : A \to B$ *a morphism* $\bar{f} : A \to \mathbf{dom}(A) \subseteq A$ *(such that a few properties hold).*
2. A **Turing object** T as some kind of coding context for programs (turing machine, register machine, finite automata, etc.)
3. A universal **application morphism** $\tau_{X,Y} : T \times X \to Y$ that represents the application of a program (in T) to data (in X) to produce a result (in Y).
4. objects defined to $\mathbb{N}$, $\mathbb{N}^2$, $\mathbb{N}^3$, and so on...
5. morphisms defined to be the codings of partial computable functions.

Yes, Turing Categories.

Yes, we're saying that if you put...

codes for machines (consistent Turing object!) as morphisms

appropriate domains and codomains as objects

enforce partiality with a restriction structure

define $\tau(e, n) = \varphi_e(n)$

you get a category!

Yes, Turing Categories.

more interesting categories if you...

1. restrict to unary, binary, ternary functions, etc.
2. define each $f : \mathbb{N}^k \to \mathbb{N}^m$ to be a k-ary function that applies m k-ary functions in parallel. (in some cases, this somewhat reproduces linear algebra).

You can also show that much of what we know about computability ($s - m - n$, recursion, etc.) holds in this abstract setting. *In some cases, something breaks — and that can also tell us something about computability.*

References [1, 2, 3, 4]

Questions?

References i


A. Asperti and A. Ciabattoni.
Effective applicative structures.
In D. Pitt, D. E. Rydeheard, and P. Johnstone, editors, *Category Theory and Computer Science*, pages 81–95, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.


T.-D. Bradley.
What is category theory anyway?, Jan 2017.


J. Cockett and P. Hofstra.
Introduction to turing categories.
Annals of Pure and Applied Logic, 156(2):183–209, 2008.


T. Leinster.
Basic category theory, 2016.