Applied Category Theory Tutorial (Draft)

Birzhan Moldagaliyev

April 7, 2019

Introduction

This set of notes serves as a tutorial for those who wish to learn some Applied Category Theory. I am no expert in Applied Category Theory, and the whole point of writing this tutorial is to solidify my knowledge of the field. Since Applied Category Theory uses machinery of Category Theory, we need to learn some of the fundamental concepts from Category Theory. However, one should realize how vast Category Theory is, so it might take a long time to master all concepts from Category Theory. For this reason, I adopted a rule of introducing applications of Category Theory as early as possible, even though there is a risk that the given applications might be too simple. As for latter thing, I would like to remind myself and the readers that no infant starts running straight out of the cradle, so working on easier examples first should not be seen as a waste of time, but as an investment of time. With this being said, dear friends, let us begin our journey to the land of Applied Category Theory.

Lesson 1: Introduction to Category Theory

Questions

1. What are the origins of Category Theory?
2. What is a category?
3. What is a functor?

What are the origins of Category Theory?

Category Theory was born in 1940s when two mathematicians, Eilenberg and Mac Lane, working in quite different areas of mathematics found a common pattern or theme in their works. Thus, they invented a language of category theory, where morphisms between objects play a major role. Since then there is an ongoing process of using Category Theory to discover connections from seemingly distinct areas of mathematics. This has become so common that mathematician invented the term of *categorification* to describe the process of translating some mathematical phenomena into the language of category theory.

What is a category?

A category is a collection of objects, and morphisms which satisfy certain properties.

Definition 1 (Category)**.** A category $\mathcal{C}$ consists of

1. Objects, denoted as $Ob(\mathcal{C})$.
2. Morphisms. For every $c_1, c_2 \in Ob(\mathcal{C})$, there is a corresponding collection of morphisms denoted $Hom_{\mathcal{C}}(c_1, c_2)$ or $\mathcal{C}(c_1, c_2)$.
3. Identity morphisms. For every object $c \in Ob(\mathcal{C})$, there is a specific morphism $id_c \in \mathcal{C}(c, c)$.

4. Composition of morphisms. There is composition map $\circ$, such that for every triple of objects c_1, c_2, c_3

$$\circ : \mathcal{C}(c_1, c_2) \times \mathcal{C}(c_2, c_3) \to \mathcal{C}(c_1, c_3)$$

satisfying the following properties

(a) Identity morphisms. For every $f \in \mathcal{C}(c_1, c_2)$

$$id_{c_2} \circ f = f = f \circ id_{c_1}$$

(b) Associativity of composition. For every triple of morphism $f \in \mathcal{C}(c_1, c_2), g \in \mathcal{C}(c_2, c_3)$ and $h \in \mathcal{C}(c_3, c_4)$

$$h \circ (g \circ f) = (h \circ g) \circ f$$

Often composition of two morphisms is written as gf, instead of $g \circ f$, where we go through f first, followed by g.

Examples and intuitions behind categories

There are many examples of categories in mathematics as

- Set, a category of sets with functions as morphisms.
- $Vect$, a category of vector spaces with linear maps as morphisms.
- Top, a category of topological spaces with continuous maps as morphisms.

As one can see, in all of above examples, objects of categories are sets with morphisms as certain kind of functions. However, there are many examples of categories where a morphisms are not excplicit functions. For example, given any graph, nodes of the graph can thought as objects, while paths between nodes can be thought as morphisms.

Intuition behind the given categories

As a rule of thumb, one could use the following intuition for formation of categories.

- $Ob(\mathcal{C})$: (elements, structure, properties).
- Hom: Morphisms maps elements of one objects into elements of another object. Morphisms preserve structure, while some properties might disappear along the way.

What is a functor?

Functors allow us to travel from category to another.

Definition 2 (Functor)**.** Given two categories $\mathcal{C}, \mathcal{D}$, a functor $F : \mathcal{C} \to \mathcal{D}$ is

1. a mapping between objects, $F : Ob(\mathcal{C}) \to Ob(\mathcal{D})$.
2. a mapping between morphims, i.e. for all objects $X, Y \in Ob(\mathcal{C})$

$$F : \mathcal{C}(X, Y) \to \mathcal{D}(FX, FY)$$

such that

(a) $F(id_X) = id_{FX}$, F preserves identities.

(b) $Fg \circ Ff = F(g \circ f)$, F preserves compositions.

Examples of Functors

Usual map	Functor based interpretation
monotone map	functor between preorder categories
monoid homomorphism	functor between monoids
database instance	functor from database schema to Set

Lesson 2: Linear Computations and Databases

Questions:

1. How can we think of linear computations in terms of category theory?
2. How can we think of databses in terms of category theory?

Linear Computations

Linear computations are widely used in almost all computational disciplines. Given a bunch of inputs $\{x_1, x_2, \ldots, x_n\}$ one of the easiest things anyone can do is to take a linear sum:

$$c_1x_1 + c_2x_2 + \ldots + c_nx_n = \sum_{i=1}^{n} c_ix_i$$

What makes above mapping so attractive? There are several reasons:

1. It can be quickly computed, more preciely in $O(n)$ time.
2. It is easily predictable in a sense that if x_1 goes us by 1, the above value changes by c_1.
3. It is relatively easy to solve system of linear equations, with the upper bound of $O(n^3)$ time needed.

Let us write a simple computer program which computes the above linear sum:

```
sum = 0
for i from 1 to n {
sum = sum + c_i*x_i
}
```

Now let us try to convert the idea of translating a linear map into computer program using the langauge of category theory. In the process of doing so, we are going to learn about two categories, a category of matrices, **Mat**, and a category of programs, **Prog**.

Matrices

The objects of this category are natural numbers. A morphism between objects m and n is an (m, n) matrix consisting of rational numbers (in principle we could work with any field). So if $m = 2$ and $n = 3$, then one of the morphisms between m and n could be

$$\begin{bmatrix} 3 & 0 & \frac{1}{2} \\ 4 & -1 & 1 \end{bmatrix}$$

Programs

The objects of this category are data types, such as a boolean type (**Bool**), an integer type (**Int**) and a rational number type (**Rat**). A morphism between types x and y is a program with an input type of x and an output type of y. For instance, if we recall the program for computing a linear sum, then its input type is $(\mathbf{Rat})^n$ and its output type is **Rat**, where we assume that the given program works with only rational numbers (besides, one can't store irrational numbers on a computer).

Connection

Having introduced the above two categories, we can now attempt to construct a functor between them. To do so, it is necessary for us to describe

1. A map between objects of **Mat** and **Prog**.
2. A map between morphisms of **Mat** and **Prog**.

so that compositions and identities are preserved. Let us set up a following functor $F : Mat \to Prog$.

1. $F(n) = (Rat)^n$, i.e. a number n is mapped into a type of rational n-tuples.
2. Given a (m, n) matrix M, $F(M)$ corresponds to a program realizing a matrix multplication by M, i.e. $x \to xM$, where $x \in (Rat)^n$.

Databases

Let's consider music listening patterns of people. Artists produce songs and people listen to these songs. Some music lovers have their favourite songs. How could we encode this simple fact? One way to encode this idea is to use database scheme, in our case, a scheme (or pattern) of how artists, songs and music lovers relate to each other. Let us construct a blueprint for the database tables. First we consider the tables corresponding to songs.

Songs			
Id	Title	Artist	Album
1	Sweet Dreams	Beyonce	I Am... Sasha Fierce
2	Hot n Cold	Katy Perry	One of the Boys
3	Poker Face	Lady Gaga	The Fame

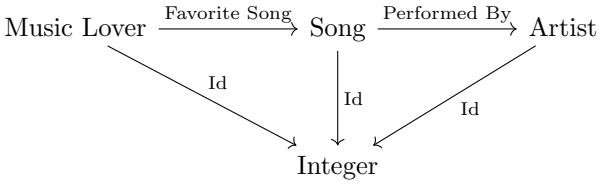
The second table provides information about music lovers.

Music Lovers		
Id	Name	Favorite Song
1	Adam Bronx	Hot n Cold
2	Amanda Hills	Sweet Dreams
3	Jacquelin Smith	Poker Face
4	Elizabeth Taylor	Sweet Dreams

The final table is devoted to artists.

Artists		
Id	Name	Style
1	Beyonce	Pop
2	Katy Pery	Pop
3	Lady Gaga	Pop

Based on this blueprint, we could visualize relationships between above entities as the following graph.



Database Instances

We have learnt that it is possible to represent relationships between database entities as a graph. By morphing arrows and extending identity arrows in such a graph, it can be transformed into a category, let's call it *database scheme category*. If it is so, then how could we represent instances of the given database? A possible way to represent instances of the given database is to construct a functor between the database scheme category and **Set**, the category of sets. In case of the presented example, the functor, F, maps the objects as follows

- $F(\text{Artist}) = \{\text{Beyonce, Katy Perry, Lady Gaga}\}$.
- $F(\text{Song}) = \{\text{Sweet Dreams, Hot n Cold, Poker Face}\}$.
- $F(\text{Music Lover}) = \{\text{Adam Bronx, Amanda Hills, Jacquelin Smith, Elizabeth Taylor}\}$.
- $F(\text{Integer}) = \{1, 2, 3\}$.

Bibliography

[1] Elaine Landry and Jean-Pierre Marquis, *Categories in Context: Historical, Foundational, and Philosophical*, Philosophia Mathematica, Volume 13, Issue 1, February 2005, Pages 143.