Intro to Category Theory: Categories

1 Intro and Motivations

Category theory is a branch of mathematics that formalizes mathematical structure. Category theory has built up quite a reputation for having quite powerful applications across a vast part of mathematics and programming languages. I imagine you have or will run into category theory concepts during your career. For example, categorical semantics, monads, functors, and proof assistants have deep connections to category theory.

Category theory is a generalization of algebra. In category theroy, we understand mathematical objects through structure preserving transformations, called morphisms. For example, even though it is possible to do so, we don't embed a single group into a category. Instead, we study groups through the category of all groups **Grp**, where objects are groups and morphisms are group homomorphisms. We then learn about groups not through the elements that make them up, which is opaque in the category, instead we understand groups relationships through morphisms.

Category theory can be used as foundation for mathematics, replacing set theory. However, we will not be taking this perspective. Though not explicit, we will actually be building the definitions of category theory on top of set theory[1]. Nevertheless, I want you to recognize the differences in the categorical style compared to the set theoretic style. In set theory we define mathematical objects by characterizing the elements that make them up. This is an *internal* view of mathematics. In category theory objects are characterized by their relationship to other objects. Often, such objects are atomic. They have no internal structure. All relevant properties are *externally* defined.

These sets of notes are not meant to be an exhaustive or superior presentation of category theory. They are just my presentation with my own quirks and motivations. There are plenty of other references if you want to learn category theory more seriously. The standard textbook is "Categories for the Working Mathematician" by Saunders Mac Lane. It's not very approachable or easy to read, but it is thorough. Mac Lane is undoubtedly, one of the founders of category theory along with Samuel Eilenberg, but they were originally topologists. Mac Lane doesn't give a lot of examples and at least for me they don't provide much intuition. A much more approachable text is the open source book "Category Theory for Programmers" by Bartosz Milewski. It's much easier to read and he provides real programming examples to illustrate category theory concepts. The downside is it is not quite as formal as a serious textbook. If you are serious about learning category theory I would recommend starting with "Category Theory for Programmers" and having a companion textbook, like "Categories for the Working Mathematician" to go to after initially absorbing a concept.

2 Categories

A category, say C, has objects and arrows/morphisms. I will usually use lower case beginning of the alphabet letters for objects $(a, b, c, ...)$ and lower case middle of the alphabet letters for arrows $(f, g, h, ...)$. I will use upper case beginning of the alphabet letters for categories $(A, B, C, ...)$.

Morphisms generalize functions and like functions they have a domain and codomain. Rather than having sets as domains and codomains, morphisms have an object as a domain and an object as a codomain. Almost always, morphisms are introduced with a function like signature to indicate their domain and codomain. For example we may introduce a morphism with $f : a \to b$ to indicate

[1]It is not necessary to build category theory on top of set theory. Category can be defined in its own way, and set theory on top of that. See elementary theory of the category of categories.

that f is a morphism and has domain a and codomain b. Often when we are talking about functions in set theory we are not so careful to write function signatures, because domains and codomains can often be inferred from context. However, in category theory we need to be much more careful. It is very important to give a signature when introducing morphisms for clarity.

2.1 Axioms

For C to be a category the following axioms must be satisfied

1. Arrows compose. If $f : a \to b$ and $g : b \to c$ are arrows of C, then there is arrow $g \circ f : a \to c$ (pronounced "g after f") in C.

2. Composition is associative. If $f : a \to b$, $g : b \to c$, and $h : c \to d$ are arrows of C, then

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

.

3. Identity arrows. Every object b has an identity arrow id_b, such that for every $f : a \to b$ and $g : b \to c$

$$id_b \circ f = f \qquad g \circ id_b = g$$

.

I want to make note of something missing from the axioms. Note that I did *not* say that a category C consists of a *set* of objects and a *set* of arrows. This is very much intentional. The collection of objects and morphisms do not need to form a set. They can be proper classes. This distinction divides categories as being either *small* or *large.*

Definition 2.1. If the collection of objects and morphisms of a category C form a set, such a category is called *small.* A category is *large* otherwise. That is if the collection of objects or morphisms is a proper class rather than a set, then C is *large.* If a category C satisfies the property that for every pair of objects a and b the collection of morphisms between a and b forms a set, then we call C *locally small.*

The reason the collection of objects doesn't have to be a set is for the same reason proper classes are defined in ZFC. It's to avoid the self referential paradoxes of naive set theory. For example, in a moment we will introduce an example category where objects are all (small) sets. With the large and small distinction we can give an answer to the question of where the collection of the objects is itself an object in this category. The answer is no, because the collection of all objects is not a set.

While many of the categories we will discuss are in fact large, we will not notice the difference. For our purposes we will not need to make much of a distinction between sets and classes.

2.2 Commutative Diagrams

The name arrow is indicative of edges in a directed graph. This is not by accident. Aside from the large/small issue categories are just directed graphs with some additional axioms. While we can almost never draw an entire category, because they are most often infinite, we can still depict finite patterns within a category using directed graphs.

Often in category we will succinctly depict properties of morphisms using a special kind of directed graph called a *commutative diagram.*

Definition 2.2. A *commutative diagram* is a directed graph where any two paths that both start at the same point and end at the same point, yield the same result.

Example 2.1. The following diagram succinctly denotes that for morphisms $f : a \to b$, $g : b \to c$, and $h : a \to c$, $g \circ f = h$.

$$\begin{tikzcd}
a \arrow[r, "f"] \arrow[dr, "h"'] & b \arrow[d, "g"] \\
 & c
\end{tikzcd}$$

To indicate a graph is a commutative diagram we might say "this diagram commutes".

Though not necessary we can redefine the axioms of a category with commutative diagrams. For all objects a, b, c and d and morphisms f, g, h, the following diagrams commute.

1. (Composition) $$\begin{tikzcd}
a \arrow[r, "f"] \arrow[dr, "g \circ f"'] & b \arrow[d, "g"] \\
 & c
\end{tikzcd}$$

2. (Associativity) $$\begin{tikzcd}
a \arrow[r, "h \circ g \circ f"] \arrow[d, "f"'] \arrow[dr, "g \circ f"] & d \\
b \arrow[r, "g"'] \arrow[ur, "h \circ g"] & c \arrow[u, "h"']
\end{tikzcd}$$

3. (Unit) $$\begin{tikzcd}
a \arrow[r, "f"] \arrow[dr, "f"'] & b \arrow[d, "id_b"] \arrow[dr, "g"] & \\
 & b \arrow[r, "g"'] & c
\end{tikzcd}$$

These diagrams succinctly capture the signatures of all the morphisms and properties they must satisfy.

2.3 Examples of Categories

Now that we have the very basics out of the way let's look at some examples.

2.3.1 Toy Example

Here is a toy example without being totally trivial. This is just a toy example, it will not have further reference outside of this section. Consider the category depicted by the following commutative diagram.

$$\begin{tikzcd}
a \arrow[loop above, "id_a"] \arrow[r, bend right, "f"'] & b \arrow[loop above, "id_b"] \arrow[l, bend right, "g"']
\end{tikzcd}$$

This category consists of two objects, and four morphisms, two of them identity arrows. While this example doesn't say much general it does have a notable characteristic. Observe that $g \circ f = id_a$ and $f \circ g = id_b$. If this happens we call f and g invertible.

Definition 2.3. Let $f : a \to b$ and $g : b \to a$ be morphisms. g is the inverse of f (and vice versa) if $g \circ f = id_a$ and $f \circ g = id_b$. If a morphism f has an inverse it is an *isomorphism.* If two objects are connected by an isomorphisms, they are *isomorphic.*

In this example f and g are isomorphisms. Thus, a and b are isomorphic.

Here's a non-example of a category to make sure you're on your toes. It also illustrates a slightly annoying feature of commutative diagrams.

$$\begin{tikzcd}
a \arrow[loop above, "id_a"] \arrow[r, bend right, "f"'] & b \arrow[loop above, "id_b"] \arrow[l, bend right, "g"'] \arrow[l, "h"']
\end{tikzcd}$$

The annoying piece is the presence of parallel arrows in the diagram. If we are strictly following the definition of a commutative diagram g and h start and end at the same point, thus they are equal. However, in almost all such situations we don't mean to indicate that $g = h$. Thus, strictly speaking what is above is *not* a commutative diagram, because I want to mean $g \neq h$. But, I want the diagram to indicate that all other paths commute. Thus, what I really want this diagram to mean is that all paths which start and end at the same point are equal *except* g and h. Practically, it makes sense for $g \neq h$ because if they were equal, I wouldn't have drawn two separate arrows. This situation will pop up again. I would say that the rule is when someone says a diagram commutes they mean the diagram commutes except for parallel arrows. Parallel arrows indicate non-equal morphisms.

Anyway, given the discussion above, why can't the diagram denote a category? It is because associativity is violated.

$$\begin{aligned}
(h \circ f) \circ g &= id_a \circ g \\
&= g \\
&\neq h \\
&= h \circ id_b \\
&= h \circ (f \circ g)
\end{aligned}$$

2.3.2 More Interesting Examples

I'm going to briefly give some more relevant examples. A good exercise would be to check that the following examples satisfy the axioms, and why they satisfy the axioms:

- The prototypical example of a category is the large category where the class of objects is all (small) sets, denoted **Set**, and morphisms are total functions. Composition is function composition. For pretty much the whole presentation of category theory, we will be referencing **Set** as a source of inspiration and understanding.

- As programmers we will make a lot of connection between category theory and functional programming. To represent this relationship we will use the programming language haskell. We can think of haskell types as forming a category. We denote this category as **Hask**. Objects of **Hask** are haskell types, such as `Int`, `Bool`, etc. Morphisms are haskell functions. A couple of points about **Hask**. **Hask** is very much like **Set**. For our purposes, we can think about types as just being a label for a set. That is, the type `Int` denotes the set of integers. This connection isn't quite right, but it's good enough for our purposes. One source where the connection breaks down has to do with non-termination of functions. That is in **Set** there is no concept of termination or non-termination. This would cause a problem for the definition of composition of haskell functions. A function composed with a non-terminating function should also not-terminate. Such a situation could be remedied by equipping sets with a $\perp$ element. The other, perhaps more serious issue with **Hask** is that because haskell is a real programming language, it has some constructs which may break the axioms of a category. I don't know enough about this issue, but it may be the case that if we are super serious then we shouldn't use **Hask** as a true example. However, for our instructional purposes, we will be fine to think of **Hask** as a category like **Set**.

- Many algebraic objects form categories. Such a category has all algebraic objects as categorical objects, and the morphisms are the homomorphisms of the algebraic objects. For example **Grp** is the category where objects are all groups, and morphisms are the group homomorphisms. Similar categories exist for rings, **Ring**, abelian groups, **Ab**, etc.

- At this point you might be thinking that for categories, objects are just always sets, and morphisms are functions. Obviously, this isn't the case, but you might be asking for a non-example, so here we are. Consider a partial order $(P, \leq)$. We can consider $(P, \leq)$ a category as follows. Objects are elements of P, and if $x \leq y$ then there is a morphism from x to y in the corresponding category. This is a category due to the properties of $\leq$. This category has the property that for any two objects there is at most one morphism between them. Such a category is called *thin*. A category that does not satisfy this property is called *thick*. The other categories mentioned in this subsection are thick.

- There are another large source of examples, which I am not qualified to discuss at length. Category theory grew out of topology in the later half of the 20th century. Many of the examples and concepts you find in a category theory textbook are generalizations from topology. An example is the category **Top**, which consists of all topological spaces as objects and continuous maps as morphisms.

2.4 Special Objects

At this point categories are just a big mess of spaghetti. You could say that a lot of what is to come is to disentangle this mess. It's gonna take quite a bit of effort, but we are in a situation to define some special objects now.

Definition 2.4. An *initial object* is an object that has a $\underline{\text{unique}}$ arrow to any other object in the category.

A couple points about initial objects. First, initial objects may not exist. However, for many interesting examples they do exist. I will say what initial objects are for the examples I gave previously in a moment. Second, the restriction that there must be a unique arrow to any other object may seem odd. Why does it have to be unique? I'm going to punt on this issue at the moment. We'll talk about it more when we talk about universal properties. For now, just take it as part of the definition. Finally, initial objects may not be unique. A category can have multiple initial objects. However, they are unique up to isomorphism.

Theorem 2.1. *Let a and b be initial objects in some category C, then $a \cong b$.*

Proof. By definition, since a is an initial object there must exist a morphism $f : a \to b$. Similarly, since b is initial there must exist a morphism $g : b \to a$. What is $g \circ f : a \to a$? Since a is an initial object there is only *one* arrow from a to a. For C to be a category there must be an identity arrow from a to a. This means that $g \circ f$ has to be id_a. Similar reasoning for b yields $f \circ g = id_b$. Thus, $a \cong b$. □

A big feature of category theory is duality. Categories are essentially directed graphs. What happens if you just reverse the arrows? You get another category C^{op}. In a similar vein, you always get another definition for free in category theory by considering the definition with the arrows reversed. For example consider the dual of initial objects. You guessed it you get terminal objects.

Definition 2.5. An *terminal object* is an object for which every other object has a unique arrow ending at that object.

The same facts about initial objects apply to terminal objects.

Here is a table of initial and terminal objects for the aforementioned categories:

Category	Initial	Terminal
Set	$\{\}$	Any singleton set
Hask	`Void`	() (unit)
Partial Order	$\bot$	$\top$
Top	Empty set	Singleton space
Grp	Singleton group	Singleton group

A good exercise would be to try and reason about why these are the initial and terminal objects of these categories.

As an aside, as in the situation with **Grp** an object which is both initial and terminal is called a null or zero object. We will not further investigate null objects.