A Brief Introduction to Categories

Tom Divers

PLRG :: Bristol

October 10, 2025

“A monad is a monoid in the category of endofunctors (of a specific category)”

"A monad is a monoid in the category of endofunctors (of a specific category)"

and other such malarkey

Content Overview

- *Today:* Categories, functors, natural transformations
- Products, exponentials, CCCs
- Models of STLC

Idea is not for you to learn this stuff properly, but to get the gist of why we care

What's a category?

Categories are a way of talking about *transformations:*

- Functions between sets
- Programs between types
- Linear maps between vector spaces
- Homptopies between spaces

and many many many many other things

What's a category?

A category $\mathcal{C}$ is defined by:

What's a category?

A category $\mathcal{C}$ is defined by:

- $\mathrm{Ob}(\mathcal{C})$: The *class* of things that we're transforming. We call these **objects**. (Sometimes we just write $\mathcal{C}$)

What's a category?

A category $\mathcal{C}$ is defined by:

- $\mathrm{Ob}(\mathcal{C})$: The *class* of things that we're transforming. We call these **objects**. (Sometimes we just write $\mathcal{C}$)
- $\mathrm{Hom}\,(\mathcal{C})$: The *class* of the transformations themselves. We call these **morphisms**.

What's a category?

A category $\mathcal{C}$ is defined by:

- $\mathrm{Ob}(\mathcal{C})$: The *class* of things that we're transforming. We call these **objects**. (Sometimes we just write $\mathcal{C}$)
- $\mathrm{Hom}\,(\mathcal{C})$: The *class* of the transformations themselves. We call these **morphisms**.

Often, the category we're talking about will be inferred from context.

What's a category?

A category $\mathcal{C}$ is defined by:

- $\mathrm{Ob}(\mathcal{C})$: The *class* of things that we're transforming. We call these **objects**. (Sometimes we just write $\mathcal{C}$)
- $\mathrm{Hom}\,(\mathcal{C})$: The *class* of the transformations themselves. We call these **morphisms**.

Often, the category we're talking about will be inferred from context.

$X, Y, Z \cdots$ are objects. $f\colon X \to Y$ is a morphism from X to Y.

What's a category?

A category $\mathcal{C}$ is defined by:

- $\mathrm{Ob}(\mathcal{C})$: The *class* of things that we're transforming. We call these **objects**. (Sometimes we just write $\mathcal{C}$)
- $\mathrm{Hom}\,(\mathcal{C})$: The *class* of the transformations themselves. We call these **morphisms**.

Often, the category we're talking about will be inferred from context.

$X, Y, Z \cdots$ are objects. $f\colon X \to Y$ is a morphism from X to Y.

$\mathrm{Hom}(X, Y)$ is the *class* of all morphisms from X to Y.

What's a category?

Identity:

- For each object X, we have a morphism $\mathrm{id}_X : X \to X$

Composition:

- For morphisms $f\colon X \to Y$ and $g\colon Y \to Z$, we have another morphism $g \circ f\colon X \to Z$

What's a category?

Identity:

- For each object X, we have a morphism $\mathrm{id}_X : X \to X$

Composition:

- For morphisms $f : X \to Y$ and $g : Y \to Z$, we have another morphism $g \circ f : X \to Z$

Laws:

What's a category?

Identity:

- For each object X, we have a morphism $\mathrm{id}_X : X \to X$

Composition:

- For morphisms $f : X \to Y$ and $g : Y \to Z$, we have another morphism $g \circ f : X \to Z$

Laws:

- **Unit:** For $f : X \to Y$, we have $f = f \circ \mathrm{id}_X = \mathrm{id}_Y \circ f$
- **Associativity:** $f \circ (g \circ h) = (f \circ g) \circ h$

Some very important categories:

- $\mathbf{Set}$:
  - Objects are sets (e.g. $\{1\}, \mathbb{N}, \{f \mid f\colon \mathbb{N} \to \{a, b, c\}\}, \cdots$)
  - $\mathsf{Hom}(X, Y) = \{f \mid f\colon X \to Y\}$
  - $\mathsf{id}_X = \lambda a.a$
  - $g \circ f = \lambda a.g(f(a))$

Some very important categories:

- $\mathbf{Set}$:
  - Objects are sets (e.g. $\{1\}, \mathbb{N}, \{f \mid f\colon \mathbb{N} \to \{a, b, c\}\}, \cdots$)
  - $\mathrm{Hom}(X, Y) = \{f \mid f\colon X \to Y\}$
  - $\mathrm{id}_X = \lambda a.a$
  - $g \circ f = \lambda a.g(f(a))$
- $\mathtt{Hask}$:
  - Objects are types in Haskell (e.g. `Int`, `Bool -> IO String`, `...`)
  - $\mathrm{Hom}(\mathtt{X}, \mathtt{Y}) = \{\mathtt{f} \mid \mathtt{f\ ::\ X\ ->\ Y}\}$
  - $\mathrm{id}_{\mathtt{X}} = \mathtt{\backslash a.a\ ::\ X\ ->\ X}$
  - $\mathtt{g} \circ \mathtt{f} = \mathtt{g.f} = \mathtt{\backslash a.g\ (f\ a)}$

Some very important categories:

- $\mathbf{Set}$:
  - Objects are sets (e.g. $\{1\}, \mathbb{N}, \{f \mid f : \mathbb{N} \to \{a, b, c\}\}, \cdots$)
  - $\mathrm{Hom}(X, Y) = \{f \mid f : X \to Y\}$
  - $\mathrm{id}_X = \lambda a.a$
  - $g \circ f = \lambda a.g(f(a))$
- $\mathtt{Hask}$:
  - Objects are types in Haskell (e.g. `Int`, `Bool -> IO String`, `...`)
  - $\mathrm{Hom}(\mathtt{X}, \mathtt{Y}) = \{\mathtt{f} \mid \mathtt{f :: X -> Y}\}$
  - $\mathrm{id}_{\mathtt{X}} =$ `\a.a :: X -> X`
  - $\mathtt{g} \circ \mathtt{f} = \mathtt{g.f} =$ `\a.g (f a)`

We'll talk about why/how these are similar in another advanced haskell

Some useful categories:

- $\mathcal{C}^{\mathsf{op}}$: Same objects as $\mathcal{C}$. If $f\colon X \to Y \in \mathcal{C}$ then $f^{\mathsf{op}} : Y \to X \in \mathcal{C}^{\mathsf{op}}$

Some useful categories:

- $\mathcal{C}^{\mathsf{op}}$: Same objects as $\mathcal{C}$. If $f\colon X \to Y \in \mathcal{C}$ then $f^{\mathsf{op}} : Y \to X \in \mathcal{C}^{\mathsf{op}}$
- **Orders:** A category where $\mathrm{Hom}(X, Y) = \emptyset$ or $\leq_{XY} : X \to Y$ is the only element of $\mathrm{Hom}(X, Y)$.

Some useful categories:

- $\mathcal{C}^{\mathsf{op}}$: Same objects as $\mathcal{C}$. If $f\colon X \to Y \in \mathcal{C}$ then $f^{\mathsf{op}} : Y \to X \in \mathcal{C}^{\mathsf{op}}$
- **Orders:** A category where $\mathrm{Hom}(X, Y) = \emptyset$ or $\leq_{XY} : X \to Y$ is the only element of $\mathrm{Hom}(X, Y)$.
- **Monoids:** One object M, $\mathrm{Hom}(M, M)$ are all the monoid elements, and the monoid operation is $\circ$.

Some useful categories:

- $\mathcal{C}^{\text{op}}$: Same objects as $\mathcal{C}$. If $f: X \to Y \in \mathcal{C}$ then $f^{\text{op}}: Y \to X \in \mathcal{C}^{\text{op}}$
- **Orders:** A category where $\text{Hom}(X, Y) = \emptyset$ or $\leq_{XY}: X \to Y$ is the only element of $\text{Hom}(X, Y)$.
- **Monoids:** One object M, $\text{Hom}(M, M)$ are all the monoid elements, and the monoid operation is $\circ$.
- **Groups:** A monoid where each morphism f is an *isomorphism* (has an inverse f^{-1}, with $f^{-1} \circ f = \text{id}_M$). Any category where each morphism has such an inverse is called a *groupoid*.

Some useful categories:

- $\mathcal{C}^{\mathsf{op}}$: Same objects as $\mathcal{C}$. If $f\colon X \to Y \in \mathcal{C}$ then $f^{\mathsf{op}} : Y \to X \in \mathcal{C}^{\mathsf{op}}$
- **Orders:** A category where $\mathsf{Hom}(X, Y) = \emptyset$ or $\leq_{XY} : X \to Y$ is the only element of $\mathsf{Hom}(X, Y)$.
- **Monoids:** One object M, $\mathsf{Hom}(M, M)$ are all the monoid elements, and the monoid operation is $\circ$.
- **Groups:** A monoid where each morphism f is an *isomorphism* (has an inverse f^{-1}, with $f^{-1} \circ f = \mathsf{id}_M$). Any category where each morphism has such an inverse is called a *groupoid*.

If you can redefine something as a category, you can often *generalise* useful things about it

Functors

It'd be nice if we could relate categories to each other.

Functors

It'd be nice if we could relate categories to each other.

Let $\mathcal{C}$ and $\mathcal{D}$ be categories. A **functor** $F : \mathcal{C} \to \mathcal{D}$ maps $X, f \in \mathcal{C}$ to $F(X), F(f) \in \mathcal{D}$.

Functors

It'd be nice if we could relate categories to each other.

Let $\mathcal{C}$ and $\mathcal{D}$ be categories. A **functor** $F : \mathcal{C} \to \mathcal{D}$ maps $X, f \in \mathcal{C}$ to $F(X), F(f) \in \mathcal{D}$.

Rules:

- If $f : X \to Y$ then $F(f) : F(X) \to F(Y)$
- $F(\mathrm{id}_X) = \mathrm{id}_{F(X)}$
- $F(f \circ g) = F(f) \circ F(g)$

Functors

It'd be nice if we could relate categories to each other.

Let $\mathcal{C}$ and $\mathcal{D}$ be categories. A **functor** $F : \mathcal{C} \to \mathcal{D}$ maps $X, f \in \mathcal{C}$ to $F(X), F(f) \in \mathcal{D}$.

Rules:

- If $f : X \to Y$ then $F(f) : F(X) \to F(Y)$
- $F(\mathsf{id}_X) = \mathsf{id}_{F(X)}$
- $F(f \circ g) = F(f) \circ F(g)$

An **endofunctor** is a functor $F : \mathcal{C} \to \mathcal{C}$.

Functors

```
-- f :: * -> * is any type constructor
-- f transforms our objects

class Functor f where
    -- fmap transforms our morphisms
    fmap :: (a -> b) -> (f a -> f b)

    -- fmap must satisfy:
        -- fmap id = id
        -- fmap (f . g) = fmap f . fmap g
```

A Haskell functor $\mathtt{f}$ is an *endofunctor* $\mathtt{f} : \mathtt{Hask} \to \mathtt{Hask}$.

What about relating functors to each other?
Let's say we want to show that `Maybe` can be simulated by `List`:

```
data Maybe a = Nothing | Just a
mmap :: (a -> b) -> (Maybe a -> Maybe b)
mmap = fmap

data List a  = Nil      | Cons a (List a)
lmap :: (a -> b) -> (List a -> List b)
lmap = fmap
```

So we need to show that

- $\exists$ `eta :: forall a. Maybe a -> List a`
- `eta . (mmap f) = (fmap f) . eta` for all `f :: a -> b`