Category Theory: Foundations

Hype for Types

1 Categories

What *is* a category?

Definition 1.1 (Category)**.** A category $\mathbb{C}$ consists of:

- A collection $\mathbb{C}_0$ (or $\mathsf{ob}(\mathbb{C})$) of "objects".
- A collection $\mathbb{C}_1$ (or $\mathsf{arr}(\mathbb{C})$) of "arrows"/"morphisms".

such that:

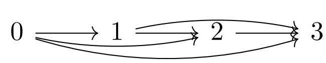
- Every arrow $u \in \mathbb{C}_1$ has a "domain"/"source" object and a "codomain"/"target" object. If u has source X and target Y, we write $f : X \to Y$.
- For every object $X \in \mathbb{C}_0$, there exists an identity arrow $\mathsf{id}_X : X \to X$.
- For arrows $f : X \to Y$ and $g : Y \to Z$, there exists an arrow $X \to Z$ which we denote $g \circ f$.
- For all $f : X \to Y$, $f \circ \mathsf{id}_X = f = \mathsf{id}_Y \circ f$.
- For all $f : W \to X, g : X \to Y, h : Y \to Z$, $(h \circ g) \circ f = h \circ (g \circ f)$.

Example 1.1 (Preorder Categories)**.** For any set S with a preorder $\leq$, we can consider a category where the objects are elements of S and there is a single morphism $\star_{x,y} : x \to y$ if $x \leq y$, for $x, y \in S$.

- Notice that every object x has an identity arrow $\mathsf{id}_x = \star_{x,x} : x \to x$, by reflexivity.
- Also, if $\star_{x,y} : x \to y$ (i.e., $x \leq y$) and $\star_{y,z} : y \to z$ (i.e., $y \leq z$), then we have an arrow $\star_{x,z} : x \to z$ (i.e., $x \leq z$) by transitivity.

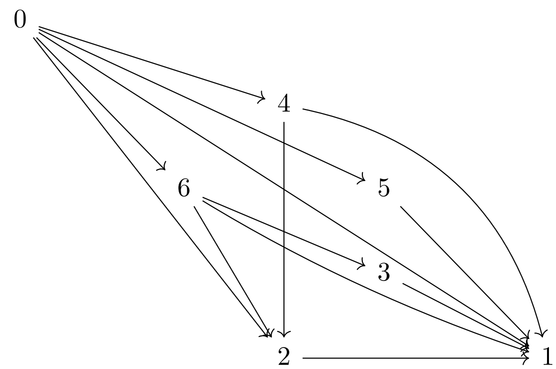
There are many concrete examples:

- Consider $\mathbb{N}$ with the standard ordering; call this category $(\mathbb{N}, \leq)$.

  We omit identity arrows for brevity.
- Similarly, consider $\mathbb{R}_{\geq 0}$ (the non-negative real numbers) with the standard ordering; call this category $(\mathbb{R}_{\geq 0}, \leq)$.
- Consider $\mathbb{N}$ with the ordering $a \leq b$ iff a is a multiple of b; call this category **Multiple**. This is reflexive (every a is a multiple of itself) and transitive (if a is a multiple of b and b is a multiple of c, then a is

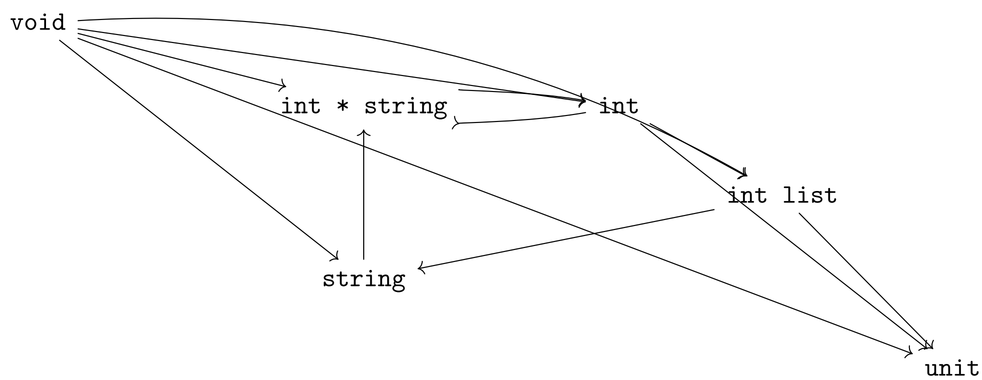
a multiple of c). Notice that every number is a multiple of 1 and 0 is a multiple of every number.



Example 1.2 (The Category **Set**)**.** The category **Set** has objects being sets and arrows being functions. Identity arrows are the identity functions, and composition is function composition.

Example 1.3 (The Category **SML**)**.** The category **SML** has objects being SML types and arrows being (total) functions. Identity arrows are the identity functions `fn (x : t) => x`, and composition is function composition via `o`.

In the following diagram, many objects arrows are omitted - there are infinitely many!



Example 1.4 (The Category **CLogic**)**.** The category **CLogic** has objects being propositions in constructive logic and arrows being present if an implication is true.

Definition 1.2 (Isomorphism)**.** Let $\mathbb{C}$ be a category.

If X_1 and X_2 are objects and $f_1 : X_1 \to X_2$ and $f_2 : X_2 \to X_1$ are arrows with $f_2 \circ f_1 = \mathsf{id}_{X_1}$ and $f_1 \circ f_2 = \mathsf{id}_{X_2}$, then we say that X_1 and X_2 are *isomorphic* objects.

2 Terminal and Initial Objects

Some categories have objects which are particularly "special" - we'll consider two special kinds of objects.

2.1 Terminal Object

Definition 2.1 (Terminal Object)**.** Let $\mathbb{C}$ be a category. A *terminal object* in $\mathbb{C}$ is an object T which satisfies the following property:

for all objects T',
there exists a unique arrow $h : T' \to T$.

Pictorally:

$$\begin{array}{c} T' \\ \Big\downarrow{\scriptstyle \exists! h} \\ T \end{array}$$

Theorem 2.1 (Uniqueness of Terminal Object)**.** *Let $\mathbb{C}$ be a category. If T and T' are both terminal objects, then T and T' are isomorphic.*

So, it is reasonable to talk about *the* terminal object, since it is unique up to isomorphism.

Example 2.1. Recall the category $(\mathbb{N}, \leq)$ from Example 1.1. There is no terminal object. Suppose n were the terminal object: then, it must be the case that for all m, $m \leq n$, but there is no such number n.

Example 2.2. Recall the category **Multiple** from Example 1.1. Here, 1 is the terminal object, since for all other objects n, there exists an arrow $n \to 1$ (since every number is a multiple of 1). Since there is at most one arrow between any two objects in **Multiple**, this arrow is unique.

Example 2.3. Recall the category **Set** from Example 1.2. Here, $\{42\}$ is the terminal object, since for all other objects S, there exists a unique arrow $S \to \{42\}$, the function mapping all inputs to 42.

There are other terminal objects, like $\{43\}$, $\{a\}$, and $\{\{\}\}$, but these are isomorphic by Theorem 2.1.

Example 2.4. Recall the category **SML** from Example 1.3. Here, `unit` is the terminal object, since for all other objects `t`, there exists a unique arrow `t -> unit`, the function mapping all inputs to `()`.

There are other terminal objects, like `unit * unit`, `void + unit`, and `datatype foo = Foo`, but these are isomorphic by Theorem 2.1.

Example 2.5. Recall the category **CLogic** from Example 1.4. Here, $\top$ (the always-true proposition) is the terminal object, since for all other objects φ, there exists a unique arrow $\varphi \implies \top$, since every proposition φ implies trivial truth.

Remark 2.6. Notice that terminal objects store only "trivial" data. Thus, we often call the terminal object of a category 1 or $1_{\mathbb{C}}$.

2.2 Initial Object

Definition 2.2 (Initial Object)**.** Let $\mathbb{C}$ be a category. A *initial object* in $\mathbb{C}$ is an object I which satisfies the following property:

for all objects I',
there exists a unique arrow $h : I \to I'$.

Pictorally:

$$\begin{array}{c} I' \\ {\scriptstyle \exists! h}\Big\uparrow \\ I \end{array}$$

Theorem 2.2 (Uniqueness of Initial Object)**.** *Let $\mathbb{C}$ be a category. If I and I' are both initial objects, then I and I' are isomorphic.*

So, it is reasonable to talk about *the* initial object, since it is unique up to isomorphism.

Example 2.7. Recall the category $(\mathbb{N}, \leq)$ from Example 1.1. The initial object is 0, since for all n, we have an arrow $0 \to n$ since $0 \leq n$. Since there is at most one arrow between two numbers, the arrow is unique.

Example 2.8. Recall the category **Multiple** from Example 1.1. The initial object is 0, since for all n, we have an arrow $0 \to n$ since 0 is a multiple of n. Since there is at most one arrow between two numbers, the arrow is unique.

Example 2.9. Recall the category **Set** from Example 1.2. Here, $\varnothing$ is the initial object, since for all other objects S, there exists a unique arrow $\varnothing \to S$.[1]

Example 2.10. Recall the category **SML** from Example 1.3. Here, `void` is the initial object, since for all other objects `t`, there exists a unique arrow `void -> t`.

Example 2.11. Recall the category **CLogic** from Example 1.4. Here, $\bot$ (the always-false proposition) is the initial object, since for all other objects φ, there exists a unique arrow $\bot \implies \varphi$, since falsity implies any proposition φ.

3 Products and Coproducts (Sums)

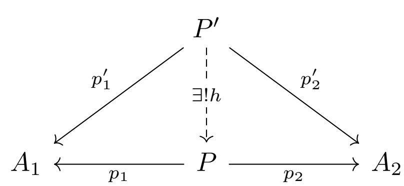
Now, let's consider how objects interact.

3.1 Product

Definition 3.1 (Product)**.** Let $\mathbb{C}$ be a category, and let A_1, A_2 be objects in $\mathbb{C}$. The *product* of A_1 and A_2 is some object P with has arrows $p_1 : P \to A_1$, $p_2 : P \to A_2$ which satisfy the following property:

for all objects P' with arrows $p_1' : P' \to A_1$, $p_2' : P' \to A_2$,
there exists a unique arrow $h : P' \to P$ such that $p_1' = p_1 \circ h$ and $p_2' = p_2 \circ h$.

Pictorally:



Theorem 3.1 (Uniqueness of Products)**.** *Let $\mathbb{C}$ be a category, and let A_1, A_2 be objects in $\mathbb{C}$. If P and P' are both products of A_1 and A_2, then P and P' are isomorphic.*

So, it is reasonable to talk about *the* product of A_1 and A_2.

Example 3.1. Recall the category $(\mathbb{N}, \leq)$ from Example 1.1, and consider objects n_1, n_2. The product of n_1 and n_2 in this category would be some object m such that $m \leq n_1$ and $m \leq n_2$, with:

for all m' with $m' \leq n_1$ and $m' \leq n_2$, then $m' \leq m$

In other words, m must be the largest number which is less than or equal to n_1 and n_2. So, m must be $\min(n_1, n_2)$.

Example 3.2. Recall the category **Multiple** from Example 1.1, and consider objects n_1, n_2. The product of n_1 and n_2 in this category would be some object m such that m is a multiple of n_1 and n_2, with:

for all m' with m' being a multiple of n_1 and n_2, m' is a multiple of m

[1] This may feel non-obvious! For example, let $S = \mathbb{N}$; then, wouldn't $f = x \mapsto 42$ and $g = x \mapsto 43$ be different arrows? In fact, they are the same function, since "for all $x \in \varnothing$, $f(x) = g(x)$" is true (vacuously; there are no such $x \in \varnothing$).

In other words, m must be the smallest number which is a multiple of n_1 and n_2. So, m must be $\text{lcm}(n_1, n_2)$.

Example 3.3. Recall the category **Set** from Example 1.2, and consider objects A, B. The product of A and B in this category would be some object P with arrows $p_A : P \to A$ and $p_B : P \to B$, with:

> for all P' with arrows $p'_A : P' \to A$ and $p'_B : P' \to B$, there exists a unique arrow $h : P' \to P$ such that $p'_A = p_A \circ h$ and $p'_B = p_B \circ h$.

Consider $P = A \times B$, the Cartesian product of our sets A and B, with $p_A = (a, b) \mapsto a$ and $p_B = (a, b) \mapsto b$. Then, given an arbitrary P', p'_A, p'_B, the unique arrow $h : P' \to P$ is $x \mapsto (p'_A(x), p'_B(x))$.

Example 3.4. Recall the category **SML** from Example 1.3, and consider types `a`, `b`. The product of `a` and `b` in this category would be some object `p` with arrows `pa : p -> a` and `pb : p -> b`, with:

> for all `p'` with arrows `pa' : p' -> a` and `pb' : p' -> b`, there exists a unique arrow `h : p' -> p` such that `pa' = pa o h` and `pb' = pb o h`.

Consider `p = a * b`, the tuple/product type of `a` and `b`, with `pa = fst = fn (a, b) => a` and `pb = snd = fn (a, b) => b`. Then, given an arbitrary `p'`, `pa'`, `pb'`, the unique arrow `h : p' -> p` is `fn x => (pa' x, pb' x)`.

Remark 3.5. Notice that not only is `int * string` a product of `int` and `string`, but in fact so are `string * int`, `string * int * unit`, and `(string * unit) + void`. This is okay, though, since products are unique up to isomorphisms, as described in Theorem 3.1.

Example 3.6. Recall the category **CLogic** from Example 1.4. We claim that the product of two propositions φ_1 and φ_2 will be $\varphi_1 \wedge \varphi_2$; the justification is left as an exercise to the reader.

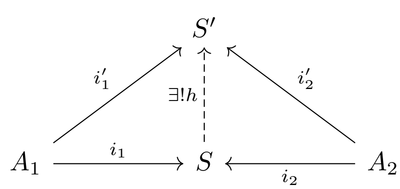
3.2 Coproducts (Sums)

We can dualize the definition of products to get *coproducts*, or *sums*.

Definition 3.2 (Coproduct)**.** Let $\mathbb{C}$ be a category, and let A_1, A_2 be objects in $\mathbb{C}$. The *coproduct* (or sum) of A_1 and A_2 is some object S with has arrows $i_1 : A_1 \to S$, $i_2 : A_2 \to S$ which satisfy the following property:

> for all objects S' with arrows $i'_1 : A_1 \to S$, $i'_2 : A_2 \to S$,
> there exists a unique arrow $h : S \to S'$ such that $i'_1 = i_1 \circ h$ and $i'_2 = i_2 \circ h$.

Pictorally:



Theorem 3.2 (Uniqueness of Coproducts)**.** *Let $\mathbb{C}$ be a category, and let A_1, A_2 be objects in $\mathbb{C}$. If S and S' are both coproducts of A_1 and A_2, then S and S' are isomorphic.*

So, it is reasonable to talk about *the* coproduct of A_1 and A_2.

Example 3.7. Recall the category $(\mathbb{N}, \leq)$ from Example 1.1, and consider objects n_1, n_2. The coproduct of n_1 and n_2 in this category would be some object m such that $n_1 \leq m$ and $n_2 \leq m$, with:

> for all m' with $n_1 \leq m'$ and $n_2 \leq m'$, then $m \leq m'$

In other words, m must be the smallest number which is greater than or equal to n_1 and n_2. So, m must be $\max(n_1, n_2)$.

Example 3.8. Recall the category **Multiple** from Example 1.1, and consider objects n_1, n_2. The coproduct of n_1 and n_2 in this category would be some object m such that n_1 and n_2 are multiples of m, with:

for all m' with n_1 and n_2 being multiples of m', m is a multiple of m'

In other words, m must be the largest number which n_1 and n_2 are both multiples of. So, m must be $\gcd(n_1, n_2)$.

Example 3.9. Recall the category **Set** from Example 1.2, and consider objects A, B. We claim that $A \uplus B$, the disjoint union of A and B, is the coproduct of A and B.[2] The justification is left as an exercise to the reader.

Example 3.10. Recall the category **SML** from Example 1.3, and consider types `a`, `b`. The coproduct of `a` and `b` in this category would be some object `s` with arrows `ia : a -> s` and `ib : b -> s`, with:

for all `s'` with arrows `ia' : a -> s'` and `ib' : b -> s'`, there exists a unique arrow `h : s -> s'` such that `ia' = ia o h` and `ib' = ib o h`.

Consider `s = a + b = (a, b) either`, the sum type of `a` and `b`, with `ia = Left` and `ib = Right`. Then, given an arbitrary `s'`, `ia'`, `ib'`, the unique arrow `h : s -> s'` is `fn Left a => ia' a | Right b => ib' b`.

Example 3.11. Recall the category **CLogic** from Example 1.4. We claim that the coproduct of two propositions φ_1 and φ_2 will be $\varphi_1 \vee \varphi_2$; the justification is left as an exercise to the reader.

4 Functors

We've considered some examples of categories and considered interesting examples of objects in a category. However, we may consider what it would mean to move between categories. For this purpose, we'll use *functors*.

Definition 4.1 (Functor)**.** A functor $F : \mathbb{C} \to \mathbb{D}$ consists of:

- a map $F_0 : \mathbb{C}_0 \to \mathbb{D}_0$
- a map $F_1 : \mathbb{C}_1 \to \mathbb{D}_1$

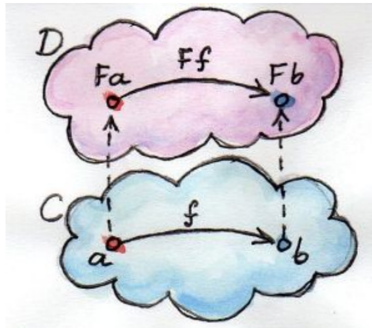
such that:

- the source and target of arrows are preserved; if $f : X \to Y$ is an arrow in $\mathbb{C}$, then $F_1(f) : F_0(X) \to F_0(Y)$ in $\mathbb{D}$.
- for every object $X \in \mathbb{C}_0$, $F_1(\mathsf{id}_X) = \mathsf{id}_{F_0(X)}$
- for arrows $f : X \to Y$ and $g : Y \to Z$, $F_1(g \circ f) = F_1(g) \circ F_1(f)$

Notice that the conditions enforce that F_0 and F_1 preserve the structure of a category, as given in Definition 1.1.

Pictorally, using an image from Bartosz Milewski:

[2] Consider: which condition would $A \cup B$ violate?



Note that the image overloads F to refer to both F_0 and F_1.

Example 4.1 (Doubling Functor on $\mathbb{N}$)**.** We can define a functor $F : (\mathbb{N}, \leq) \to (\mathbb{N}, \leq)$ as follows:

$$F_0(x) = 2x$$
$$F_1(\star_{x,y}) = \star_{2x,2y}$$

The map F_1 is well defined because if $x \leq y$, then $2x \leq 2y$.

Since there is at most one arrow per pair of objects (namely, $\star$), the functor laws are trivially satisfied.

Example 4.2 (Floor Functor)**.** We can define a functor $F : (\mathbb{R}_{\geq 0}, \leq) \to (\mathbb{N}, \leq)$ as follows:

$$F_0(x) = \lfloor x/2 \rfloor$$
$$F_1(\star_{x,y}) = \star_{\lfloor x/2 \rfloor, \lfloor y/2 \rfloor}$$

The map F_1 is well defined because if $x \leq y$, then $\lfloor x/2 \rfloor \leq \lfloor y/2 \rfloor$.

Since there is at most one arrow per pair of objects (namely, $\star$), the functor laws are trivially satisfied.

Remark 4.3. Any monotone function between posets induces a functor, where "f is monotone" is defined as "$x \leq y$ implies $f(x) \leq f(y)$".

Example 4.4 (List Functor)**.** We can define a functor $\mathsf{List} : \mathbf{SML} \to \mathbf{SML}$ as follows:

$$F_0(\texttt{t}) = \texttt{t list}$$
$$F_1(\texttt{f}) = \texttt{List.map f}$$

Observe that the functor laws are satisfied:

$$\texttt{List.map Fn.id} = \texttt{Fn.id}$$
$$\texttt{List.map (g o f)} = \texttt{List.map g o List.map f}$$

Remark 4.5. Many common types form functors: `'a * 'a`, `'a option`, `'a tree`, `'a shrub`, `'a stream`, `int -> 'a`, and so on.

Remark 4.6. Not all polymorphic types form functors: consider `'a t = 'a -> int`, and try to write `map : ('a -> 'b) -> 'a t -> 'b t`.

4.1 Connection to SML

Restricting our attention to category **SML**, we can define a signature which describes functors **SML** $\to$ **SML**. So as not to confuse ourself with the word `functor` used in the SML module system, we use the word `MAPPABLE`.

```
signature MAPPABLE =
  sig
    type 'a t
    val map : ('a -> 'b) -> 'a t -> 'b t

    (* Invariants (functor laws):
      map id = id
      map (g o f) = map g o map f
    *)
  end
```

Of course, we cannot enforce the functor laws via SML types, so we include them as commented "invariants".

Defining a functor now involves implementing a structure ascribing to `MAPPABLE`:

```
structure ListMappable : MAPPABLE =
  struct
    type 'a t = 'a list
    val map = List.map
  end
```

In fact, many category theoretic ideas will be useful abstractions in functional programming!