

# Category Theory Formalization in Agda

EAFIT - UdeA seminar

José Ramírez-Gómez

Universidad EAFIT

May 2025

## What is Category Theory?

Category theory is a branch of mathematics that:

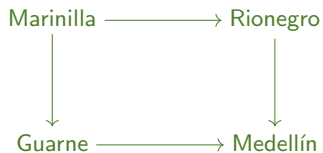
- Studies connections between different ideas.
- Allows us to understand order and structures.
- Focuses on relationships.
- Helps to see under which context things are equivalent.

# Commutative Diagrams

## Idea

A commutative diagram is a diagram in which all paths that start and end at the same point determine the same result.

## Example



# What is a Category?

## Definition (Category)

A category  $\mathcal{C}$  consists of [6]:

- A collection of objects,  $\text{Obj}(\mathcal{C})$ , denoted by letters  $A, B, C$ , etc.
- A collection of arrows or morphisms,  $\text{Mor}(\mathcal{C})$ , denoted by letters  $f, g, h$ , etc.
- Two maps,  $\text{dom}, \text{cod} : \text{Mor}(\mathcal{C}) \rightarrow \text{Obj}(\mathcal{C})$ , assigning to each arrow  $f$  its domain  $\text{dom}(f)$  and codomain  $\text{cod}(f)$ . For an arrow  $f$  with domain  $A$  and codomain  $B$ , we write  $f : A \rightarrow B$ .
- For each pair of objects  $A, B$ , we define the **set**

$$\text{Hom}(A, B) := \{f \in \text{Mor}(\mathcal{C}) \mid f : A \rightarrow B\}$$

which we call a *Hom-set*.

# What is a Category?

## Definition (Category)

- For any triple of objects  $A, B, C$ , the composition of morphisms,

$$\circ : \text{Hom}(A, B) \times \text{Hom}(B, C) \rightarrow \text{Hom}(A, C).$$

Given  $f \in \text{Hom}(A, B)$  and  $g \in \text{Hom}(B, C)$ , we write  $g \circ f$  to denote the composition  $g$  after  $f$ .



- For each object  $A$ , an identity arrow,  $1_A : A \rightarrow A$ .

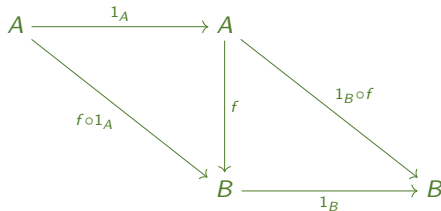
# What is a Category?

## Definition (Category)

Such that the following axioms hold:

- Identities: For any morphism  $f : A \rightarrow B$ , we have

$$f \circ \mathbf{1}_A = f = \mathbf{1}_B \circ f.$$

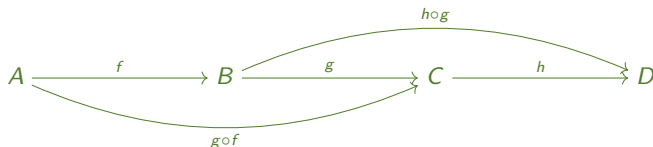


# What is a Category?

## Definition (Category)

- Associativity: For any morphisms  $f : A \rightarrow B$ ,  $g : B \rightarrow C$ ,  $h : C \rightarrow D$ , we have

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



## Example: Set

Set is the category in which:

- **Objects:** Sets.
- **Arrows:** Functions between sets.
- **Identity arrow:** Is the identity function

$$1_X : X \rightarrow X$$

$$x \mapsto x$$

- **Composition:** Is the composition of functions. If  $f : X \rightarrow Y$  and  $g : Y \rightarrow Z$  are two functions, then

$$g \circ f : X \rightarrow Z$$

$$x \mapsto g(f(x))$$

is their composite function.

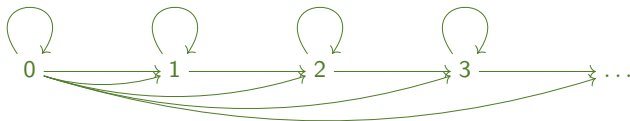


## Example: Natural Numbers

Starting from the natural numbers, we can construct a category as follows:

- **Objects:** Natural numbers  $0, 1, 2, 3, \dots$
- **Arrows:**  $A \longrightarrow B$  whenever  $A \leq B$ .
- **Identity:** Is given by the reflexivity of  $\leq$ , that is, every natural number is less than or equal to itself.
- **Composition:** Is given by the transitivity of  $\leq$ , that is, if  $a \leq b$  and  $b \leq c$  then  $a \leq c$ .

What does this category look like?



## What is Agda?

- Dependently typed programming language.
- Proof assistant.
- It is based on intuitionistic type theory.



## Agda: Basic Concepts

### Inductive definition of $\mathbb{N}$

In Agda we can define the set  $\mathbb{N}$  inductively as follows

```
1 data Nat : Set where
2   zero : Nat
3   suc  : Nat → Nat
```

### What is a term of this type?

If we want to write the number 7, we can do this

```
1 x : Nat
2 x = suc (suc (suc (suc (suc (suc (suc zero))))))
```

## Agda: Basic Concepts

### What is a term of this type?

Since writing numbers in that way is inconvenient we can use flags to make our lives easier.

```
1 {-# BUILTIN NATURAL Nat #-}  
2 y : Nat  
3 y = 7
```

### Operations on $\mathbb{N}$

Since we have defied  $\mathbb{N}$  we can define addition inductively as follows

```
1 _+_ : Nat → Nat → Nat  
2 zero + n = n  
3 (suc m) + n = suc (m + n)
```

# Agda: Basic Concepts

## Owr first proof

Lets show that  $2 + 3 = 5$

```
1      _ : 2 + 3 ≡ 5
2      _ =
3      begin
4          2 + 3
5      ≡⟨⟩      -- is shorthand for
6          (suc (suc zero)) + (suc (suc (suc zero)))
7      ≡⟨⟩      -- inductive case
8          suc ((suc zero) + (suc (suc (suc zero))))
9      ≡⟨⟩      -- inductive case
10         suc (suc (zero + (suc (suc (suc zero)))))
11      ≡⟨⟩      -- base case
12         suc (suc (suc (suc (suc zero))))
13      ≡⟨⟩      -- is longhand for
14         5
15      □
```

## Formalization: Category

Now that we have all the background needed, we are able to start formalizing things. First of all we need to define what is a category in Agda

```
1  record Category : Set1 where
2
3  field
4      Obj : Set
5      Hom : Obj → Obj → Set
6
7      id   : ∀ {A} → Hom A A
8      comp : ∀ {A B C} → Hom A B → Hom B C → Hom A C
9
10     assoc : ∀ {A B C D} (f : Hom A B) (g : Hom B C) (h : Hom C D) →
11             comp f (comp g h) ≡ (comp (comp f g) h)
12     idL   : ∀ {A B} (f : Hom A B) → comp id f ≡ f
13     idR   : ∀ {A B} (f : Hom A B) → comp f id ≡ f
```

## Formalization: First Example

### Category of $\mathbb{N}$ and $\leq$

Previously we saw that there is a category whose objects are the natural numbers  $\mathbb{N}$  and the arrows are given by the relation  $\leq$ . First of all we need to define what does it mean for a natural number to be less than or equal to other natural number.

```
1      variable
2      k l m n : Nat
3
4      data _≤_ : Nat → Nat → Set where
5      ≤-zero :          zero ≤ n
6      ≤-suc  : m ≤ n → suc m ≤ suc n
```

## Formalization: First Example

### Category of $\mathbb{N}$ and $\leq$

Recall that in this category the identity arrows are given by the reflexivity of the relation  $\leq$  so

```
1  ≤-refl : {n : Nat} → n ≤ n
2  ≤-refl {n = zero}   = ≤-zero
3  ≤-refl {n = suc k}  = ≤-suc ≤-refl
```

Composition is given by the transitivity of the relation  $\leq$  so

```
1  ≤-trans : {k l m : Nat} → k ≤ l → l ≤ m → k ≤ m
2  ≤-trans ≤-zero l≤m           = ≤-zero
3  ≤-trans (≤-suc k≤l) (≤-suc l≤m) = ≤-suc (≤-trans k≤l l≤m)
```



## Formalization: First Example

We already have the objects and the arrows of the category. What we must do now is prove that the axioms of a category are satisfied.

### Left and right identity

```
1  -- left identity law
2  ≤-IdL : {m n : Nat} → (f : m ≤ n) → (≤-trans ≤-refl f) ≡ f
3  ≤-IdL ≤-zero      = refl
4  ≤-IdL (≤-suc f) = cong ≤-suc (≤-IdL f)
5
6  -- right identity law
7  ≤-IdR : {m n : Nat}(f : m ≤ n) → (≤-trans f ≤-refl) ≡ f
8  ≤-IdR ≤-zero      = refl
9  ≤-IdR (≤-suc f) = cong ≤-suc (≤-IdR f)
```

## Formalization: First Example

### Associativity

Lets show the associativity

```
1  ≤-assoc : {k l m n : Nat} → (f : k ≤ l) (g : l ≤ m) (h : m ≤ n)
2      → ≤-trans f (≤-trans g h) ≡ ≤-trans (≤-trans f g) h
3  ≤-assoc ≤-zero g h = refl
4  ≤-assoc (≤-suc f) (≤-suc g) (≤-suc h) = cong ≤-suc (≤-trans-assoc f g h)
```

## Formalization: First Example

### Putting all together

We have shown that all the needed properties hold for this category. Now we are able to fill all the gaps in the definition of a Category

```
1  natCat : Category
2  natCat = record
3      { Obj      = Nat
4        ; Hom     =  $\leq$ 
5        ; id      =  $\leq$ -refl
6        ; comp    =  $\leq$ -trans
7        ; assoc   =  $\leq$ -assoc
8        ; idL     =  $\leq$ -IdL
9        ; idR     =  $\leq$ -IdR
10     }
```

## Example: Monoids

### Definition (Monoid)

A monoid  $(M, \cdot)$  is a set  $M$  together with a binary operation

$$\cdot : M \times M \rightarrow M$$

$$(a, b) \mapsto a \cdot b$$

which satisfies the following axioms:

- **Associativity:**  $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ , for all  $a, b, c \in M$ .
- **Identity existence:** there is a unique  $e \in M$  such that  $a \cdot e = a = e \cdot a$ , for all  $a \in M$ .

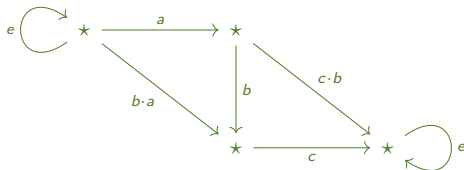
## Example: Monoids

A monoid  $(M, \cdot)$  can be seen as a category with one object. Let  $\mathcal{M}$  be a category defined by:

- There is just one object, say  $\star$ .
- Any element  $a \in M$  is an arrow  $a : \star \rightarrow \star$  in  $\mathcal{M}$ .
- Composition of arrows is the operation of the monoid, that is, if  $a, b \in \text{Ar}(\mathcal{M})$  then  $a \circ b = a \cdot b$ .
- The identity arrow  $1_\star$  is defined to be the monoid identity  $e$ .

What does this category look like?

If  $a, b, c \in M$  then



## Example: Monoids

$$(\mathbb{N}, +)$$

The category associated with this monoid is that one where:

- The object is  $\star$ .
- The arrows are the natural numbers:  $0, 1, 2, \dots$
- The identity arrow is  $0$ .
- The composition of arrows is given by the addition of natural numbers  $+$ .

$$(\mathbb{N}, \cdot)$$

The category associated with this monoid is that one where:

- The object is  $\star$ .
- The arrows are the natural numbers:  $0, 1, 2, \dots$
- The identity arrow is  $1$ .
- The composition of arrows is given by the multiplication of natural numbers  $\cdot$ .

# Formalization: Monoid as a Category

## Definition of monoid

We now formalize the definition of a monoid. This serves as a first step toward interpreting monoids as categories.

```
1      record Monoid : Set1 where
2          field
3              Carrier : Set
4              *_      : Carrier → Carrier → Carrier
5              ε       : Carrier
6
7          field
8              monAssoc : ∀ {x y z} → x * (y * z) ≡ (x * y) * z
9              monIdL   : ∀ {x} → ε * x ≡ x
10             monIdR    : ∀ {x} → x * ε ≡ x
```

## Formalization: Monoid as a Category

Now the data of the category

### Objects and morphisms

```
1      mObj : ∀ (M : Monoid) → Set
2      mObj M = T

3
4      mHom : ∀ (M : Monoid) → T → T → Set
5      mHom M _ _ = Carrier M
```

### Composition and identity

```
1      mComp : ∀ (M : Monoid) → (a b : Carrier M) → Carrier M
2      mComp M = *_ M

3
4      mId : ∀ (M : Monoid) → Carrier M
5      mId M = ε M
```



## Formalization: Monoid as a Category

And the axioms

### Left and right identity

```
1      mIdL : ∀ (M : Monoid) → (a : Carrier M) → *_ M (mId M) a ≡ a
2      mIdL M f = monIdL M
3
4      mIdR : ∀ (M : Monoid) → (a : Carrier M) → *_ M a (mId M) ≡ a
5      mIdR M a = monIdR M
```

### Associativity

```
1      mAssoc : ∀ (M : Monoid) → (a b c : Carrier M)
2              → *_ M a (*_ M b c) ≡ *_ M (*_ M a b) c
3      mAssoc M a b c = monAssoc M
```

## Formalization: Monoid as a Category

### Putting all together

```
1  cat :  $\forall$  (M : Monoid)  $\rightarrow$  Category
2  cat M = record
3      { Obj      = mObj
4        ; Hom     = mHom
5        ; id      = mId
6        ; comp    = _*_ M
7        ; assoc   = mAssoc
8        ; idL     = mIdL
9        ; idR     = mIdR
10     }
```

## Other Examples

- **Vect**: Objects are vector spaces and morphisms are linear transformations.
- **Pos**: Objects are partially ordered sets and morphisms are monotonic functions.
- **Top**: Objects are topological spaces and morphisms are continuous maps.
- **Grp**: Objects are groups and morphisms are group homomorphisms.

## What is a functor?

Functors are the notion of morphisms between categories.

### Definition (Functor)

Let  $\mathcal{C}$  and  $\mathcal{D}$  be two categories. A functor  $F : \mathcal{C} \rightarrow \mathcal{D}$  consists of two morphisms:

- **Object morphism:**  $F_0 : \text{Obj}(\mathcal{C}) \rightarrow \text{Obj}(\mathcal{D})$ , which assigns to each object  $A \in \mathcal{C}$  an object  $F_0(A) \in \mathcal{D}$ .
- **Arrow morphism:**  $F_1 : \text{Ar}(\mathcal{C}) \rightarrow \text{Ar}(\mathcal{D})$ , which assigns to each morphism  $f : A \rightarrow B$  in  $\mathcal{C}$  a morphism  $F_1(f) : F_0(A) \rightarrow F_0(B)$  in  $\mathcal{D}$ .

These must satisfy the following conditions:

- **Identity preservation:**  $F_1(1_A) = 1_{F_0(A)}$  for all objects  $A$  in  $\mathcal{C}$ .
- **Composition preservation:**  $F_1(g \circ f) = F_1(g) \circ F_1(f)$  for all composable morphisms  $f : A \rightarrow B$ ,  $g : B \rightarrow C$  in  $\mathcal{C}$ .

## Formalization: Functor

The last formalization we present is that of a functor

### Functor

```
1      record Functor (C1 D1 : Category) : Set1 where
2
3      private
4        module C = Category C1
5        module D = Category D1
6
7      field
8        F0 : C.Obj → D.Obj
9        F1 : ∀ {A B} (f : C.Hom A B) → D.Hom (F0 A) (F0 B)
10
11        id    : ∀ {A} → C.Hom A A ≡ D.Hom (F0 A) (F0 A)
12        comp  : ∀ {A B C} (f : C.Hom A B) (g : C.Hom B C) → F1 (C.comp f g)
13                ≡ D.comp (F1 f) (F1 g)
```

## Future Work

- Formalize examples of functors.
- Prove some theorems about categories.
- Build more examples.

Thanks!

# Appendix



# Agda: Equality and Proofs

## Propositional equality $\equiv$ and constructor `refl`

In Agda, propositional equality is defined as:

```
1 data _≡_ {A : Set} (x : A) : A → Set where
2   refl : x ≡ x
```

- $x \equiv y$  is the type of proofs that  $x$  and  $y$  are equal.
- `refl` is the canonical proof that any value is equal to itself.

## Function congruence: `cong`

```
1 cong : ∀ {A B : Set} {x y : A} → (f : A → B) → x ≡ y → f x ≡ f y
```

- If  $x \equiv y$ , then applying the same function  $f$  to both sides preserves equality:  $fx \equiv fy$ .

## Code Repository

The code used in this presentation is available at the following GitHub repository:

### Formalization of Category Theory in Agda

<https://github.com/jmramirez1204/category-theory-formalization>

The repository contains:

- Definitions of basic categorical structures (categories, functors, monoids as categories).
- Examples and proofs written in Agda.
- Supporting modules for equational reasoning and types.

## References I



S. Abramsky and N. Tzevelekos.

*Introduction to Categories and Categorical Logic*, page 3–94.

Springer Berlin Heidelberg, 2010.



Agda Development Team.

Agda: A dependently typed functional programming language.

<https://github.com/agda/agda>, 2025.

Available on GitHub. Accessed: 2025-05-18.



S. Awodey.

*Category Theory*.

Oxford Logic Guides. OUP Oxford, 2010.



J. Z. Hu and J. Carette.

Formalizing category theory in agda.

In *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 327–342, 2021.

## References II



T. Leinster.

Basic category theory, 2016.



S. Mac Lane.

*Categories for the working mathematician*, volume 5.

Springer Science & Business Media, 2013.