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

$$
\begin{tikzcd}
\text{Marinilla} \arrow[r] \arrow[d] & \text{Rionegro} \arrow[d] \\
\text{Guarne} \arrow[r] & \text{Medellín}
\end{tikzcd}
$$

What is a Category?

Definition (Category)

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

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

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

which we call a *Hom-set*.

What is a Category?

Definition (Category)

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

$$\circ : \mathrm{Hom}(A, B) \times \mathrm{Hom}(B, C) \to \mathrm{Hom}(A, C).$$

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

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[rr, bend right, "g \circ f"'] & B \arrow[r, "g"] & C \end{tikzcd}$$

- For each object A, an identity arrow, $\mathbf{1}_A : A \to A$.

What is a Category?

Definition (Category)

Such that the following axioms hold:

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

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

$$\begin{tikzcd}
A \arrow[r, "1_A"] \arrow[dr, "f \circ 1_A"'] & A \arrow[d, "f"] \arrow[dr, "1_B \circ f"] & \\
& B \arrow[r, "1_B"'] & B
\end{tikzcd}$$

What is a Category?

Definition (Category)

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

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

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[rr, bend right, "g \circ f"'] & B \arrow[r, "g"] \arrow[rr, bend left, "h \circ g"] & C \arrow[r, "h"] & D
\end{tikzcd}$$

Example: Set

Set is the category in which:

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

$$
\begin{aligned}
\mathbf{1}_X : X &\to X \\
x &\mapsto x
\end{aligned}
$$

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

$$
\begin{aligned}
g \circ f : X &\to Z \\
x &\mapsto g(f(x))
\end{aligned}
$$

  is their composite function.

Example: Natural Numbers

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

- **Objects:** Natural numbers $0, 1, 2, 3, \ldots$.
- **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?

$$\begin{tikzcd}
0 \arrow[loop above] \arrow[r] \arrow[r, bend right] \arrow[rr, bend right] \arrow[rrr, bend right] \arrow[rrrr, bend right] & 1 \arrow[loop above] \arrow[r] & 2 \arrow[loop above] \arrow[r] & 3 \arrow[loop above] \arrow[r] & \ldots
\end{tikzcd}$$

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

```
data Nat : Set where
  zero : Nat
  suc  : Nat → Nat
```

What is a term of this type?

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

```
x : Nat
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 inconvinient we can use flags to make our lifes easier.

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

Operations on $\mathbb{N}$

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

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

Agda: Basic Concepts

Owr first proof

Lets show that $2 + 3 = 5$

```
_ : 2 + 3 ≡ 5
_ =
  begin
    2 + 3
  ≡⟨⟩      -- is shorthand for
    (suc (suc zero)) + (suc (suc (suc zero)))
  ≡⟨⟩      -- inductive case
    suc ((suc zero) + (suc (suc (suc zero))))
  ≡⟨⟩      -- inductive case
    suc (suc (zero + (suc (suc (suc zero)))))
  ≡⟨⟩      -- base case
    suc (suc (suc (suc (suc zero))))
  ≡⟨⟩      -- is longhand for
    5
   ∎
```

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

```
record Category : Set₁ where

  field
    Obj : Set
    Hom : Obj → Obj → Set

    id   : ∀ {A} → Hom A A
    comp : ∀ {A B C} → Hom A B → Hom B C → Hom A C

    assoc : ∀ {A B C D} (f : Hom A B) (g : Hom B C) (h : Hom C D) →
            comp f (comp g h) ≡ (comp (comp f g) h)
    idL   : ∀ {A B} (f : Hom A B) → comp id f ≡ f
    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.

```
variable
  k l m n : Nat

data _≤_ : Nat → Nat  → Set where
  ≤-zero :          zero ≤ n
  ≤-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

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

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

```
≤-trans : {k l m : Nat} → k ≤ l → l ≤ m → k ≤ m
≤-trans  ≤-zero l≤m            = ≤-zero
≤-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 satisfyed.

Left and right identity

```
-- left identity law
≤-IdL : {m n : Nat} → (f : m ≤ n) → (≤-trans ≤-refl f) ≡ f
≤-IdL ≤-zero     = refl
≤-IdL (≤-suc f) = cong ≤-suc (≤-IdL f)

-- right identity law
≤-IdR : {m n : Nat}(f : m ≤ n) →  (≤-trans f ≤-refl) ≡ f
≤-IdR ≤-zero     = refl
≤-IdR (≤-suc f) = cong ≤-suc (≤-IdR f)
```

Formalization: First Example

Associativity

Lets show the associativity

```
≤-assoc : {k l m n : Nat} → (f : k ≤ l) (g : l ≤ m ) (h : m ≤ n)
          → ≤-trans f (≤-trans g h) ≡ ≤-trans (≤-trans f g) h
≤-assoc ≤-zero g h                = refl
≤-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

```
natCat : Category
natCat = record
         { Obj   = Nat
         ; Hom   = _≤_
         ; id    = ≤-refl
         ; comp  = ≤-trans
         ; assoc = ≤-assoc
         ; idL   = ≤-IdL
         ; idR   = ≤-IdR
         }
```

Example: Monoids

Definition (Monoid)

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

$$\begin{aligned} \cdot : M \times M &\to M \\ (a, b) &\mapsto a \cdot b \end{aligned}$$

which satisifes 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 $a : \star \to \star$ in $\mathcal{M}$.
- Composition of arrows is the operation of the monoid, that is, if $a, b \in \mathrm{Ar}(\mathcal{M})$ 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

$$
\begin{tikzcd}
\star \arrow[loop left, "e"] \arrow[r, "a"] \arrow[dr, "b\cdot a"'] & \star \arrow[d, "b"] \arrow[dr, "c\cdot b"] & \\
& \star \arrow[r, "c"'] & \star \arrow[loop right, "e"]
\end{tikzcd}
$$

Example: Monoids

$(\mathbb{N}, +)$

The category associated with this monoid is that one where:

- The object is $\star$.
- The arrows are the natural numbers: $0, 1, 2, \ldots$.
- 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: $0, 1, 2, \ldots$.
- 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.

```
record Monoid : Set₁ where
  field
    Carrier : Set
    _*_     : Carrier → Carrier → Carrier
    ε       : Carrier

  field
    monAssoc : ∀ {x y z} → x * (y * z) ≡ (x * y) * z
    monIdL   : ∀ {x} → ε * x ≡ x
    monIdR   : ∀ {x} → x * ε ≡ x
```

Formalization: Monoid as a Category

Now the data of the category

Objects and morphisms

```
mObj : ∀ (M : Monoid) → Set
mObj M = ⊤

mHom : ∀ (M : Monoid) → ⊤ → ⊤ → Set
mHom M _ _ = Carrier M
```

Composition and identity

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

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

Formalization: Monoid as a Category

And the axioms

Left and right identity

```
mIdL : ∀ (M : Monoid) → (a : Carrier M) → _*_ M (mId M) a ≡ a
mIdL M f = monIdL M

mIdR : ∀ (M : Monoid) → (a : Carrier M) → _*_ M a (mId M) ≡ a
mIdR M a = monIdR M
```

Associativity

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

Formalization: Monoid as a Category

Putting all together

```
        cat : ∀ (M : Monoid) → Category
        cat M = record
              { Obj   = mObj
              ; Hom   = mHom
              ; id    = mId
              ; comp  = _*_ M
              ; assoc = mAssoc
              ; idL   = mIdL
              ; idR   = mIdR
              }
```

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 $F : \mathcal{C} \to \mathcal{D}$ consists of two morphisms:

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

These must satisfy the following conditions:

- **Identity preservation:** $F_1(\mathbf{1}_A) = \mathbf{1}_{F_0(A)}$ 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 $f : A \to B$, $g : B \to C$ in $\mathcal{C}$.

Formalization: Functor

The last formalization we present is that of a functor

Functor

```
record Functor (C1 D1 : Category) : Set1 where

  private
   module C = Category C1
   module D = Category D1

  field
    F0 : C.Obj → D.Obj
    F1 : ∀ {A B} (f : C.Hom A B) → D.Hom (F0 A) (F0 B)

    id   : ∀ {A} → C.Hom A A ≡ D.Hom (F0 A) (F0 A)
    comp : ∀ {A B C} (f : C.Hom A B) (g : C.Hom B C) → F1 (C.comp f g)
           ≡ 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:

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

```
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.