Formalising the Density Theorem

Piyush Vasudha Naresh
Imperial College London
piyush.naresh24@imperial.ac.uk

March 2025

Abstract

We present a formalisation of Yoneda's Lemma and the Density Theorem in Lean. The latter states that every sheaf is the co-limit of representable sheaves. Interestingly, our proof and formalisation of the Density Theorem does not make use of Yoneda's Lemma.

1 Introduction

In this project we will first formalise Yoneda's lemma in Lean. We will define the functor $Hom(A, \_) : \mathcal{C} \longrightarrow Sets$, and the Yoneda embedding $y : \mathcal{C}^{op} \longrightarrow Fun(\mathcal{C}, Sets)$ from scratch. Then we will construct a bijection between $Hom(A, B)$ and $Hom(Hom(A, \_), Hom(B, \_))$, for all $A, B \in \mathcal{C}^{op}$, and use this to show that y is full and faithful. All this will be a refinement of project 2.

Next, we will formalise the fact that every sheaf is the co-limit of representable sheaves. More explicitly, given any functor $P : \mathcal{C} \longrightarrow Sets$, we will define a functor $A : (\int_{\mathcal{C}} P)^{op} \longrightarrow \mathcal{C}^{op}$ such that $P = lim_{\rightarrow}(y \circ A)$. This result is sometimes called the Density Theorem.

There are many ways to prove this theorem. We will take the most direct approach and define a tuple (P, ϕ), and show that this gives a co-cone and, in fact, a co-limit of F. Unlike most proofs of the Density Theorem[1], this approach will not use the fact that the Yoneda embedding y is fully faithful.

In the next section, we will present a proof of the Density Theorem as described above. Although we also formalise Yoneda's lemma, to minimise repetition, we will not present a proof of it here.

Next, we will present a detailed explanation of the code along with screenshots. To conclude, we will reflect on the challenges encountered and the key insights gained during this project.

2 Theory

Let $\mathcal{C}$ be a locally small category, by which we mean $Hom(A, B)$ is a set for all $A, B \in obj(\mathcal{C})$.

Let $P : \mathcal{C} \longrightarrow Sets$ be a functor. So, P is an object in $Fun(\mathcal{C}, Sets)$, which is the category of functors from $\mathcal{C}$ to $Sets$, with morphisms as natural transformations between functors. The objects of $Fun(\mathcal{C}, Sets)$ are also called sheaves.

2.1 The Yoneda Embedding

Fix an object A in $\mathcal{C}$.
Then we can define the functor $Hom(A, \_) : \mathcal{C} \longrightarrow Sets$ as:
$X \mapsto Hom(A, X)$, for all $X \in obj(\mathcal{C})$, and
for all $X, Y \in obj(\mathcal{C})$, the function $Hom(A, \_) : Hom(X, Y) \longrightarrow Hom(Hom(A, X), Hom(A, Y))$ is defined as:
$(f : X \longrightarrow Y)$ goes to the map taking $(g : A \longrightarrow X)$ to $(f \circ g : A \longrightarrow Y)$.
So, $Hom(A, \_)(f) = f \circ \_$.

Now, we can define the Yoneda embedding $y : \mathcal{C}^{op} \longrightarrow Fun(\mathcal{C}, Sets)$ as:
$A \mapsto Hom(A, \_)$, for all $A \in obj(\mathcal{C}^{op})$, and
for all $A, B \in obj(\mathcal{C}^{op})$, the function $y : Hom(B, A) \longrightarrow Hom(Hom(B, \_), Hom(A, \_))$ is defined as:
$f : B \longrightarrow A$ goes to the natural transformation $y(f) : Hom(B, \_) \longrightarrow Hom(A, \_)$, which is defined as:
for all $X \in obj(\mathcal{C})$, $y(f)_X : Hom(B, X) \longrightarrow Hom(A, X)$, where $y(f)_X(g) = g \circ f$ (Note: this makes sense because f goes from A to B in $\mathcal{C}$).
So, $y(f)_X = \_ \circ f$.
The fact that $y(f)$ is a natural transformation easily follows from the definitions and the fact that composition is associative.

Similarly, it is easy to check that $y : \mathcal{C}^{op} \longrightarrow Fun(\mathcal{C}, Sets)$ is a functor.

2.2 The Category of Elements of P

This is the category $J = \int_{\mathcal{C}} P$, defined as:
The objects of J are of the form (B, y), where B is an object in $\mathcal{C}$ and $y \in P(B)$. Note that $P(B)$ is a set.
The morphisms from (A, x) to (B, y) are the morphisms $u : A \longrightarrow B$ in $\mathcal{C}$ such that $P(u)(x) = y$.

There is the obvious projection functor $\pi : J \longrightarrow \mathcal{C}$ defined as:
$(B, y) \mapsto B$, and
$(u : (A, x) \longrightarrow (B, y)) \mapsto (u : A \longrightarrow B)$.

For the purposes of this exposition we define the following modification of π:

Definition 2.2.1. $A : J^{op} \longrightarrow \mathcal{C}^{op}$, *where:*
$X \mapsto \pi(X)$, *and*
$(u : Y \longrightarrow X) \mapsto (A(u) : A(Y) \longrightarrow A(X))$, *where* $u : X \longrightarrow Y$ *in* J, *and* $A(u)$ *corresponds to* $\pi(u) : \pi(X) \longrightarrow \pi(Y)$ *in* $\mathcal{C}$.

2.3 P as the Co-limit of $y \circ A$

Now, we can define the functor $F = y \circ A : J^{op} \longrightarrow Fun(\mathcal{C}, Sets)$.
So, for any $(C, p) \in obj(J^{op})$, $F : (C, p) \mapsto Hom(C, \_)$.

We claim $P = lim_{\rightarrow J^{op}} F$.
So, we need a tuple (P, ϕ), where ϕ is a collection of morphisms in $Fun(\mathcal{C}, Sets)$ indexed by the objects of J^{op}:

Definition 2.3.1. *For any* $X = (C, p) \in obj(J^{op})$, *define the morphism* $\phi_X : F(X) \longrightarrow P$ *as:*
for any $B \in obj(\mathcal{C})$, $(\phi_X)_B : Hom(C, B) \longrightarrow P(B)$ *is defined as:*
$(f : C \longrightarrow B) \mapsto P(f)(p)$.

Lemma 2.3.1. *For each object* $X = (C, p)$ *in* J^{op}, *the morphism* $\phi_X : F(X) \longrightarrow P$, *as described above, is a natural transformation.*

Proof. Suppose $g : D \longrightarrow E$ in $\mathcal{C}$.
Then, for any $f \in Hom(C, D)$ we have:

$(P(g) \circ (\phi_X)_D)(f) = P(g)(P(f)(p)) = (P(g) \circ P(f))(p) = P(g \circ f)(p) = ((\phi_X)_E \circ (g \circ \_))(f)$.
Note that $F(X)(g) = Hom(C, \_)(g) = g \circ \_$.
Hence, the required square commutes.

$\square$

Before we can show that (P, ϕ) is the co-limit of F, we have to show that it is a co-cone of F.

Lemma 2.3.2. *(P, ϕ) is a co-cone of $F : J^{op} \longrightarrow Fun(\mathcal{C}, Sets)$.*

Proof. Suppose $u : Y \longrightarrow X$ in J^{op}, where $Y = (C^{'}, p^{'})$ and $X = (C, p)$.
So, $u : C \longrightarrow C^{'}$ in $\mathcal{C}$, and $P(u)(p) = p^{'}$.

We have to show that $\phi_X \circ F(u) = \phi_Y$:
So, for any $B \in obj(\mathcal{C})$, we have to show that $(\phi_X)_B \circ F(X)_B = (\phi_Y)_B$.
Let $f \in Hom(C^{'}, B)$. By unfolding definitions we get:
$((\phi_X)_B \circ F(X)_B)(f) = (\phi_X)_B(y(u)_B(f)) = (\phi_X)_B(f \circ u) = P(f \circ u)(p) = (P(f) \circ P(u))(p) = P(f)(P(u)(p)) = P(f)(p^{'}) = (\phi_Y)_B(f)$.

Hence, (P, ϕ) is a co-cone of F.

$\square$

Now, we are ready to show that (P, ϕ) is the co-limit of F.

Theorem 2.3.1. *(P, ϕ) is the co-limit of $F : J^{op} \longrightarrow Fun(\mathcal{C}, Sets)$.*

Proof. Let (N, ψ) be any co-cone of F.
We claim that there is a unique morphism $u : P \longrightarrow N$ such that, for all $X \in obj(J^{op})$, we have $u \circ \phi_X = \psi_X$.
Note that a morphism in $Fun(\mathcal{C}, Sets)$ is a natural transformation.

We define the morphism $u : P \longrightarrow N$ as follows:
For any $B \in obj(\mathcal{C})$, define $u_B : P(B) \longrightarrow N(B)$ as:
For any $x \in P(B)$, consider $Z = (B, x) \in obj(J^{op})$. Now, $\psi_Z : F(Z) \longrightarrow N$ is a natural transformation. So, we have the map $(\psi_Z)_B : Hom(B, B) \longrightarrow N(B)$.

Define $u_B(x) = (\psi_Z)_B(id_B)$.

Claim: $u : P \longrightarrow N$, as defined above, is a natural transformation.
Consider any $g : A \longrightarrow B$ in $\mathcal{C}$.
We have to show that $N(g) \circ u_A = u_B \circ P(g)$.
For any $x \in P(A)$, $(N(g) \circ u_A)(x) = N(g)((\psi_{(A,x)})_A(id_A))$, and $(u_B \circ P(g))(x) = u_B(P(g)(x)) = (\psi_{(B,y)})_B(id_B)$, where $y = P(g)(x)$.
Now, from the naturality of $\psi_X : F(X) \longrightarrow N$, for all $X \in obj(J^{op})$, we get:
$N(g)((\psi_{(A,x)})_A(id_A)) = (\psi_{(A,x)})_B(g \circ id_A) = (\psi_{(A,x)})_B(g)$.
Also, for $g : (A,x) \longrightarrow (B,y)$ in J, using the fact that (N, ψ) is a co-cone of F, we get:
$\psi_{(A,x)} \circ y(g) = \psi_{(B,y)}$.
So, looking at the B component, and at the value id_B, we get:
$(\psi_{(B,y)})_B(id_B) = (\psi_{(A,x)})_B(id_B \circ g) = (\psi_{(A,x)})_B(g)$.
Hence, the claim is true.

Claim: for all $X = (C,p) \in obj(J^{op})$, we have $u \circ \phi_X = \psi_X$.
So, for all $B \in obj(\mathcal{C})$ we need to check that $u_B \circ (\phi_X)_B = (\psi_X)_B$.
For any $f : C \longrightarrow B$ in $\mathcal{C}$, we have:
$(u_B \circ (\phi_X)_B)(f) = u_B(P(f)(p)) = (\psi_{(B,q)})_B(id_B)$, where $q = P(f)(p)$.
And, $(\psi_X)_B(f) = (\psi_{(C,p)})_B(f)$.
Now, for $f : (C,p) \longrightarrow (B,q)$ in J, using the fact that (N, ψ) is a co-cone of F, we get:
$\psi_{(C,p)} \circ y(f) = \psi_{(B,q)}$.
So, looking at the B component, and at the value id_B, we get:
$(\psi_{(B,q)})_B(id_B) = (\psi_{(C,p)})_B(id_B \circ f) = (\psi_{(C,p)})_B(f)$.
Hence, the claim is true.

Finally, we claim that $u : P \longrightarrow N$ is the unique natural transformation with this property.
Suppose $u' : P \longrightarrow N$ is a natural transformation such that, for all $X \in obj(J^{op})$, we have $u' \circ \phi_X = \psi_X$.
So, we have to show that for all $B \in obj(\mathcal{C})$, $u_B = u'_B : P(B) \longrightarrow N(B)$.
For any $x \in P(B)$, $u_B(x) = (\psi_{(B,x)})_B(id_B)$.
Also, we know that at $X = (B,x)$ and for the B component, $u'_B \circ (\phi_X)_B = (\psi_X)_B$.
Therefore, $u_B(x) = (\psi_X)_B(id_B) = u'_B((\phi_X)_B(id_B)) = u'_B(P(id_B)(x)) = u'_B(id_{P(B)}(x)) = u'_B(x)$.
Hence, $u = u'$, and the claim is true.

Hence, (P, ϕ) is the co-limit of the functor $F : J^{op} \longrightarrow Fun(\mathcal{C}, Sets)$.

$\square$

Remark: Given the functor $F : J^{op} \longrightarrow Fun(\mathcal{C}, Sets)$, we can form the category of co-cones of F, with morphisms $v : (N, \psi) \longrightarrow (M, \eta)$, such that $v : N \longrightarrow M$ in $Fun(\mathcal{C}, Sets)$, with $v \circ \psi_X = \eta_X$, for all $X \in obj(J^{op})$.
In this language, the above theorem states that for any co-cone (N, ψ) of F, there is a unique morphism $u : (P, \phi) \longrightarrow (N, \psi)$.
Hence, (P, ϕ) is an initial object in the category of co-cones of F, i.e., it is the co-limit of F.

2.4 Summary

Suppose $\mathcal{C}$ is a locally small category.

Let $y : \mathcal{C}^{op} \longrightarrow Fun(\mathcal{C}, Sets)$ be the Yoneda embedding, defined as:
$A \mapsto Hom(A, \_)$.

Let $P : \mathcal{C} \longrightarrow Sets$ be a functor, and let $J = \int_{\mathcal{C}} P$ be the category of elements of P.

We defined the functor $F : J^{op} \longrightarrow Fun(\mathcal{C}, Sets)$ as $y \circ A$, where $A : J^{op} \longrightarrow \mathcal{C}^{op}$ is the modification of the projection functor (as described in definition 2.2.1).
So, $F : (C, p) \mapsto C \mapsto Hom(C, \_)$.

Then we constructed the tuple (P, ϕ), where for any $X \in obj(J^{op})$, we defined the natural transformation $\phi_X : F(X) \longrightarrow P$ as in definition 2.3.1.

Finally, in lemma 2.3.2 we showed that (P, ϕ) is the co-cone of F, and then in theorem 2.3.1 we showed that it is the co-limit of F.
Hence, $P = lim_{\rightarrow J^{op}}(y \circ A)$.

3 Formalisation in Lean

Warning: In Lean, given $f : A \longrightarrow B$ and $g : B \longrightarrow C$, the composition is written as $f \circ g : A \longrightarrow C$.

Code Snippet 1:

```
import Mathlib.Tactic
import Mathlib.CategoryTheory.Category.Basic
import Mathlib.CategoryTheory.Functor.Basic
import Mathlib.CategoryTheory.NatTrans
import Mathlib.CategoryTheory.Functor.Category
import Mathlib.CategoryTheory.Types
import Mathlib.CategoryTheory.Elements
import Mathlib.CategoryTheory.Limits.Cones

set_option autoImplicit false
set_option relaxedAutoImplicit false

open CategoryTheory Category Limits


universe u

-- C is a category with objects and Hom classes of Type u.
variable (C : Type u) [Category.{u} C]

variable{C}

/--
Defining the covariant functor Hom(A,_) from C to the category of Sets,
where A is a fixed object in C.
-/
@[simps]
def Hom_Functor (A : C) : C ⥤ Type u where

  -- Maps object X in C to the set Hom(A,X)
  obj := fun X  => A ⟶ X

  -- For objects X and Y in C, it gives a function from Hom(X,Y) to Hom(Hom(A,X), Hom(A,Y)) defined as,
  -- (f : X ⟶ Y) goes to the map taking (g : A ⟶ X) to (g ≫ f : A ⟶ Y)
  map := fun f => fun g => g ≫ f

```

Explanation 1:

First, we import all the required category theory modules from Mathlib. Then we define the category $\mathcal{C}$, with which we will be working throughout the project. $\mathcal{C}$ has objects and Hom classes of type u. Finally, for a fixed object A in $\mathcal{C}$, we define the functor $Hom(A, \_) : \mathcal{C} \implies \mathit{Type}\ u$, as in project 2.

Code Snippet 2:

```lean
/--
y is the Yoneda embedding, going from Cᵒᵖ to Fun(C, Sets).
-/
@[simps]
def y : Cᵒᵖ ⥤ (C ⥤ Type u) where

  -- Maps object A in Cᵒᵖ to the functor Hom(A,_)
  obj := fun A => Hom_Functor (A.unop)

  -- For A and B in Cᵒᵖ, y gives a map from Hom(B, A) to Hom(Hom(B, _), Hom(A, _)) defined as,
  -- (f : B ⟶ A) goes to the natural transformation η : Hom(B, _) ⟶ Hom(A, _), where η.app X is the map
  -- taking (g : B ⟶ X) to (f ≫ g).
  -- Note: this makes sense if you remember that in C, f goes from A to B.
  map := fun {B A} f => {

    -- Defining each component as specified above
    app := fun X => fun g => f.unop ≫ g

    -- Proving that this is a natural transformation.
    -- Just unravel the def of Hom(B, _) and Hom(A, _) and use associativity
    naturality := by
      intro U V f1
      ext h
      have p1 : (Hom_Functor B.unop).map f1 h = h ≫ f1 := rfl
      have p2 : (Hom_Functor A.unop).map f1 (f.unop ≫ h) = (f.unop ≫ h) ≫ f1 := rfl
      simp only [types_comp_apply]
      rw [p1, p2, assoc f.unop h f1]

  }

```

Explanation 2:

We define the Yoneda embedding $y : \mathcal{C}^{op} \implies (\mathcal{C} \implies \mathit{Type}\ u)$, as in project 2.

Code Snippet 3:

```
/--
Now, Yoneda's Lemma states that this functor y is fully faithful, i.e.
For any A and B in Cᵒᵖ the map from Hom(B, A) to Hom(Hom(B, _), Hom(A, _)) is bijective.
For any fixed A and B, we will show this by constructing an inverse.
-/
@[simps]
def Yoneda_bij (B A : Cᵒᵖ) : (B ⟶ A) ≅ (y.obj B ⟶ y.obj A) := {

  -- The map from Hom(B, A) to Hom(Hom(B, _), Hom(A, _)) is given by ψ
  hom := fun f => y.map f

  -- The inverse is defined as follows:
  -- For any natural transformation η from Hom(B, _) to Hom(A, _), map it to (η.app B)(𝟙 B).
  inv := fun η => (η.app B.unop (𝟙 B.unop)).op

  -- Showing hom ≫ inv is the identity on Hom(B, A).
  -- Just need to unravel the def of ψ(g).app B, which is a function from Hom(B, B) to Hom(A, B),
  -- and apply it at 𝟙 B.
  hom_inv_id := by
   ext g
   simp only [types_comp_apply, types_id_apply]
   have q1 : (y.map g).app B.unop = fun h => g.unop ≫ h := rfl
   rw [q1]
   simp

  -- Showing inv ≫ hom is the identity on Hom(Hom(B, _), Hom(A, _)).
  -- Just need to unravel the def of the natural transformation ψ(η.app B (𝟙 B)), where η is in Hom(Hom(B, _), Hom(A, _)).
  -- Then show it is the same as η by showing they give the same morphism for all X in C.
  inv_hom_id := by
   ext η X a
   simp only [types_comp_apply, types_id_apply]
   have q2 : (y.map (η.app B.unop (𝟙 B.unop)).op).app X = fun g => η.app B.unop (𝟙 B.unop) ≫ g := rfl
   rw [q2]
   simp

}
```

Explanation 3:

For fixed A and B in $\mathcal{C}^{op}$, we define a bijection called "$Yoneda\_bij$", between $Hom(B,A)$ and $Hom(Hom(B,\_), Hom(A,\_))$, as in project 2.

Code Snippet 4:

```
/--
Using Yoneda_bij to show y is full.
-/
@[simps]
def Yoneda_Full : Full (y : Cᵒᵖ ⥤ (C ⥤ Type u)) := {

  preimage := fun {A B : Cᵒᵖ} {α : y.obj A ⟶ y.obj B} => (Yoneda_bij A B).inv α

}

/--
Using Yoneda_bij to show y is faithful.
-/
theorem Yoneda_Faithful : Faithful (y : Cᵒᵖ ⥤ (C ⥤ Type u)) := {

  map_injective := by

    intro X Y
    rw [Function.Injective]
    intro f1 f2 h
    have p1 : (Yoneda_bij X Y).inv (y.map f1) = (Yoneda_bij X Y).inv (y.map f2) := congrArg (Yoneda_bij X Y).inv h

    have p2 : (Yoneda_bij X Y).inv ((Yoneda_bij X Y).hom f1) = f1 := hom_inv_id_apply (Yoneda_bij X Y) f1
    have p3 : (Yoneda_bij X Y).hom f1 = y.map f1 := rfl
    rw [p3] at p2
    rw [p2] at p1

    have p4 : (Yoneda_bij X Y).inv ((Yoneda_bij X Y).hom f2) = f2 := hom_inv_id_apply (Yoneda_bij X Y) f2
    have p5 : (Yoneda_bij X Y).hom f2 = y.map f2 := rfl
    rw [p5] at p4
    rw [p4] at p1

    exact p1

}
```

Explanation 4:

We use "$Yoneda\_bij$" to show that the functor y is full and faithful. We deviate from project 2 by making use of the in-built classes "$Full$" and "$Faithful$".

Code Snippet 5:

```
-- The above was a refinement of Project 2.
-- The goal of this section is to show that (P : C ⥤ Type u) is the co-limit of A ⋙ y, for some functor A.
-- This is the well known and extremely useful result that says: every sheaf is the co-limit of representable sheafs.

-- Functor from C to Type u
variable (P : C ⥤ Type u)


/--
There is a projection functor π : P.Elements ⥤ C, defined as (C, p) goes to C.
A is defined as the obvious modification of π going from P.Elementsᵒᵖ to Cᵒᵖ.
-/
@[simps]
def A : P.Elementsᵒᵖ ⥤ Cᵒᵖ where

    obj := fun X => Opposite.op (X.unop).1

    map := fun f => Opposite.op (f.unop).val



/--
The functor F is defined as the composition of A and the Yoneda embedding y.
-/
@[simps!]
def F : P.Elementsᵒᵖ ⥤ (C ⥤ Type u) := (A P) ⋙ y

```

Explanation 5:

Now we start to prove the Density Theorem. We define the functors $P : \mathcal{C} \implies Type\ u$, $A : P.Elements^{op} \implies \mathcal{C}^{op}$ (as in definition 2.2.1), and $F = A \circ y$. The goal is to show that P is a co-limit of F.

Code Snippet 6:

```
-- Now we will show that P is the colimit of F.

/--
To do this we need a tuple (P, φ), where for each object X = (C, p) in P.Elementsᵒᵖ,
the natural transformation φ_X : F(X) = Hom(C,_) ⟶ P, is defined as follows:
for all B ∈ obj(C), define (φ_X)_B : Hom(C,B) ⟶ P(B) as f goes to P(f)(p).
-/
@[simps]
def φ (X : P.Elementsᵒᵖ) : (F P).obj X ⟶ P where

    -- Defined as above.
    app := fun B => fun f => (P.map f) (X.unop).2

    -- Follows easily by unfolding definitions and using the fact that P is a functor.
    naturality := by
      intro D E g
      ext f
      simp only [types_comp_apply]
      have p1 : (((F P).obj X).map g f) = f ≫ g := rfl
      rw [p1, Functor.map_comp P f g]
      rfl

```

Explanation 6:

For each object X in $P.Elements^{op}$, we define the morphism $\phi_X : F(X) \longrightarrow P$, as in definition

2.3.1. As part of the definition we prove the naturality of ϕ_X as in lemma 2.3.1. Now we have the tuple (P, ϕ) as a candidate for the co-limit of F.

Code Snippet 7:

```
-- First, we claim that (P, φ) is co-cone of F.
-- That is, for every u : Y ⟶ X in P.Elementsᵒᵖ, we have F(u) ≫ φ_X = φ_Y.

/--
Using Lean's in-built co-cone structure we show that (P,φ) is a co-cone of F.
-/
@[simps]
def CoconePφ : Cocone (F P) where

  pt := P -- Apex object P.

  -- Natural transformation ι : F ⟶ P_c, such that for all X in P.Elementsᵒᵖ, ι.app X = φ_X.
  -- Here P_c is the constant functor mapping to P.
  -- The naturality of ι is equivalent to co-cone condition for (P,φ).
  ι := {

    app := (φ P)

    -- Follows by unfolding definitions, using the fact that P is a functor, and using the fact that
    -- for all v : (A,x) ⟶ (B,y) in P.Elements, P(v)(x) = y.
    naturality := by
      intro Y X v
      simp only [Functor.const_obj_obj, Functor.const_obj_map, comp_id]
      have q1 : (F P).map v = y.map ((v.unop).val).op := rfl
      rw [q1]
      ext B q2
      simp only [FunctorToTypes.comp]
      have q3 : (y.map ((v.unop).val).op).app B = fun f => (v.unop).val ≫ f := rfl
      have q4 : (φ P X).app B = fun f => (P.map f) (X.unop).2 := rfl
      have q5 : (φ P Y).app B = fun f => (P.map f) (Y.unop).2 := rfl
      rw [q3, q4, q5]
      simp only [FunctorToTypes.map_comp_apply]
      have q6 : P.map (v.unop).val (X.unop).2 = (Y.unop).2 := ↑v.unop.2
      rw [q6]

    }

```

Explanation 7:

We prove that (P, ϕ) is a co-cone of F. We use the in-built class "*Cocone*", which takes as input the functor F, and has fields "*pt*" and "τ", as explained below:

"*pt*" is the apex object, which is P in this case.

The natural transformation $\tau : F \longrightarrow P_c$, where $P_c : P.Elements^{op} \longrightarrow Fun(\mathcal{C}, Sets)$ is the constant functor mapping to P, is defined as:

$\tau_X = \phi_X : F(X) \longrightarrow P$, for all objects X in $P.Elements^{op}$.

It is clear that the naturality of τ is equivalent to the co-cone condition, and is proved exactly as in lemma 2.3.2.

Code Snippet 8:

```
/--
Lemma L1 will need while defining the co-limit object.
It states that for g : A ⟶ B, and co-cone s = (N, ψ) of F, we have:
N(g)(ψ_(A,x))_A (𝟙 A) = (ψ_(A,x))_B (𝟙 A ≫ g), where x ∈ P(A).
This follows by unfolding definitions and from the naturality of ψ_X : F(X) ⟶ N, for all X ∈ P.Elementsᵒᵖ.
-/
lemma L1 (A B : C) (g : A ⟶ B) (s : Cocone (F P)) (x : (CoconePφ P).pt.obj A) :
    s.pt.map g ((s.ι.app (Opposite.op { fst := A, snd := x })).app A (𝟙 A)) =
    (s.ι.app (Opposite.op { fst := A, snd := x })).app B (𝟙 A ≫ g) := by

  simp only [Functor.const_obj_obj]

  let X : P.Elementsᵒᵖ := Opposite.op { fst := A, snd := x }
  let ψ_X := s.ι.app X

  have p1 : (ψ_X.app A) ≫ s.pt.map g = ((F P).obj X).map g ≫ (ψ_X.app B) := (NatTrans.naturality ψ_X g).symm

  have p2 : ((F P).obj X).map g = fun f => f ≫ g := rfl

  have p3 : (ψ_X.app A ≫ s.pt.map g) (𝟙 A) = (((F P).obj X).map g ≫ ψ_X.app B) (𝟙 A) := congrFun p1 (𝟙 A)

  rw [p2] at p3

  have p4 : s.pt.map g (ψ_X.app A (𝟙 A)) = (ψ_X.app B) (𝟙 A ≫ g) := p3

  exact p4

  done
```

Explanation 8:

We prove lemma $L1$, which states:

For $g : A \longrightarrow B$ in $\mathcal{C}$, $x \in P(A)$, and co-cone (N, ψ) of F we have:

$N(g)(\psi_{(A,x)})_A(id_A) = (\psi_{(A,x)})_B(id_A \circ g)$.

As shown in theorem 2.3.1, this follows from the naturality of $\psi_X : F(X) \longrightarrow N$, for all objects X in $P.Elements^{op}$.

$L1$ will be used later to show that (P, ϕ) is a co-limit of F.

Code Snippet 9:

```
/--
Lemma L2 we will need while defining the co-limit object.
It states that for g : A ⟶ B, and co-cone s = (N, ψ) of F, we have:
(ψ_Y)_B (𝟙 B) = (ψ_X)_B (g ≫ 𝟙 B), where X = (A,x), Y = (B,y), and y = P(g)(x).
This follows by unfolding definitions and from the fact that (N, ψ) is a co-cone of F.
-/
lemma L2 (A B : C) (g : A ⟶ B) (s : Cocone (F P)) (x : (CoconePφ P).pt.obj A) :
    (s.ι.app (Opposite.op { fst := B, snd := (CoconePφ P).pt.map g x })).app B (𝟙 B) =
    (s.ι.app (Opposite.op { fst := A, snd := x })).app B (g ≫ 𝟙 B) := by

  simp only [Functor.const_obj_obj]

  let X : P.Elementsᵒᵖ := Opposite.op { fst := A, snd := x }
  let y := (CoconePφ P).pt.map g x
  let Y : P.Elementsᵒᵖ := Opposite.op { fst := B, snd := y }
  let ψ_X := s.ι.app X
  let ψ_Y := s.ι.app Y

  have h : P.map g x = y := rfl
  let G : X.unop ⟶ Y.unop := ⟨g, h⟩

  have t1 : (F P).map (Opposite.op G) ≫ ψ_X = ψ_Y := Cocone.w s (Opposite.op G)

  have t2 : ((F P).map (Opposite.op G) ≫ ψ_X).app B = ψ_Y.app B := congrFun (congrArg NatTrans.app t1) B

  have t5 : (((F P).map (Opposite.op G)).app B ≫ ψ_X.app B) (𝟙 B) = ψ_Y.app B (𝟙 B) := congrFun t2 (𝟙 B)

  have t7 : ψ_X.app B (g ≫ (𝟙 B)) = ψ_Y.app B (𝟙 B) := t5

  exact id t7.symm

  done


/--
Lemma L3 we will need while defining the co-limit object. It is just a slightly different version of lemma L2.
-/
lemma L3 (s : Cocone (F P)) (X : P.Elementsᵒᵖ) (B : C) (f : ((F P).obj X).obj B) :
    (s.ι.app (Opposite.op { fst := B, snd := ((CoconePφ P).ι.app X).app B f })).app B (𝟙 B) =
    (s.ι.app X).app B (f ≫ 𝟙 B) := by

  simp only [Functor.const_obj_obj]
  exact L2 P (X.unop.1) B f s (X.unop.2)
  done
```

Explanation 9:

We prove lemmas $L2$ and $L3$.

$L2$ states:

For $g : A \longrightarrow B$ in $\mathcal{C}$ and co-cone (N, ψ) of F we have:

$(\psi_Y)_B(id_B) = (\psi_X)_B(g \circ id_B)$, where $X = (A, x)$, $y = P(g)(x)$, and $Y = (B, y)$.

As shown in theorem 2.3.1, this follows from the fact that (N, ψ) is a co-cone of F.

$L3$ is a slight variant of $L2$.

$L2$ and $L3$ will be used later to show that (P, ϕ) is a co-limit of F.

Code Snippet 10:

```
/--
Finally, we cliam that (P, φ) is a co-limit of F.
That is, for any co-cone s = (N,ψ) of F, there is a unique morphism u : P ⟶ N such that
for all X in P.Elementsᵒᵖ, φ_X ≫ u = ψ_X.
-/
@[simps]
def ColimitPφ : IsColimit (CoconePφ P) where

  -- Defining the natural transformation u : P ⟶ N as:
  -- u_B(x) = (ψ_(B,x))_B (𝟙 B), for all x in P(B).
  desc s := {

    app := fun B => fun x => ((s.ι).app (Opposite.op ⟨B, x⟩)).app B (𝟙 B)

    naturality := by
      intro A B g
      ext x
      simp only [Functor.const_obj_obj, types_comp_apply]

      -- From the naturality of ψ_X : F(X) ⟶ N. Proved in lemma L1.
      have q1 : s.pt.map g ((s.ι.app (Opposite.op { fst := A, snd := x })).app A (𝟙 A)) =
      (s.ι.app (Opposite.op { fst := A, snd := x })).app B (𝟙 A ≫ g) := L1 P A B g s x

      -- From the fact that (N, ψ) is a co-cone of F. Proved in lemma L2.
      have q2 : (s.ι.app (Opposite.op { fst := B, snd := (CoconePφ P).pt.map g x })).app B (𝟙 B) =
      (s.ι.app (Opposite.op { fst := A, snd := x })).app B (g ≫ 𝟙 B) := L2 P A B g s x

      rw [q1, q2]
      simp
  }
```

Explanation 10:

We start to prove that (P, ϕ) is a co-limit of F. We use the in-built class "$IsColimit$", which takes as input the co-cone defined earlier, and has fields "$desc$", "fac", and "$uniq$".

Here we define "$desc$", which asks for the morphism $u : P \longrightarrow N$, where N is the apex object of another co-cone of F. We define u and prove its naturality exactly as in theorem 2.3.1.

Code Snippet 11:

```
  -- Prove that for all X in P.Elementsᵒᵖ, φ_X ≫ u = ψ_X.
  fac := by
    intro s X
    ext B f
    simp only [Functor.const_obj_obj, FunctorToTypes.comp]

    -- From the fact that (N, ψ) is a co-cone of F. Proved in lemma L3.
    have q4 : (s.ι.app (Opposite.op { fst := B, snd := ((CoconePφ P).ι.app X).app B f })).app B (𝟙 B) =
    (s.ι.app X).app B (f ≫ 𝟙 B) := L3 P s X B f

    rw [q4]
    simp


  -- Prove that u is the unique morphism with this property.
  -- Let v : P ⟶ N be another morphism such that 'fac' holds. Claim: u = v.
  -- Follows easily from the hypothesis, and by unfolding definitions.
  uniq := by
    intro s v h1
    ext B x
    specialize h1 (Opposite.op (B, x))
    simp only [Functor.const_obj_obj]

    have q6 : ((CoconePφ P).ι.app (Opposite.op { fst := B, snd := x })).app B ≫ v.app B =
    (s.ι.app (Opposite.op { fst := B, snd := x })).app B := congrFun (congrArg NatTrans.app h1) B

    have q8 :  v.app B (((CoconePφ P).ι.app (Opposite.op { fst := B, snd := x })).app B (𝟙 B)) =
    (s.ι.app (Opposite.op { fst := B, snd := x })).app B (𝟙 B) := congrFun q6 (𝟙 B)

    have q9 : ((CoconePφ P).ι.app (Opposite.op { fst := B, snd := x })).app B (𝟙 B) =
    P.map (𝟙 B) x := rfl

    have q10 : P.map (𝟙 B) x = (𝟙 P.obj B) x := FunctorToTypes.map_id_apply P x

    have q11 : (𝟙 P.obj B) x = x := rfl

    rw [q11] at q10
    rw [q10] at q9
    rw [q9] at q8

    exact q8

```

Explanation 11:

Here we prove the "fac" and "$uniq$" fields.

"fac" asks to show that for all objects X in $P.Elements^{op}$, we have $\phi_X \circ u = \psi_X$, where (N, ψ) is any co-cone of F, and u is defined as before.

"$uniq$" asks to show that for a fixed co-cone (N, ψ) of F, u is the unique morphism with this property.

Both are proved exactly as in theorem 2.3.1.

4 Discussion

This was a very interesting project to work on. Although fairly straightforward, it was a very good exercise in category theory and Lean. Unlike most proofs of the Density Theorem, a very direct approach was taken here. Starting from the basic definitions of the category of elements, co-cones, and co-limits as universal co-cones, at each stage we took the most obvious (and sometimes not so obvious, but the only possible) step towards the goal. Surprisingly, this worked and we were able to prove and formalise the Density Theorem. Certainly, category theory is one area of mathematics where this approach usually works out.

After proving the theorem, it was noticed that Yoneda's lemma (the fact that y is fully faithful) was not used anywhere. This is very interesting because all other, less direct proofs of the Density Theorem use Yoneda's lemma[1]. Although quite tedious, this is one benefit of taking the direct approach.

As in project 2, we chose to set both the objects and Hom classes of $\mathcal{C}$ to type u. This was done so that Lean did not need to decide on the sizes of various universes. This could also have been avoided by specifying exactly which universe is larger, but that solution would have been too "fiddly".

This project uses many in-built structures from Mathlib like "$Full$", "$Faithful$", "_.*Elements*", "*Cocone*", and "$IsColimit$". This certainly made things much simpler and is good practice in general since it allows us to use a host of associated results already in Mathlib. It is also interesting to note that the proof (on paper) and formalisation (in Lean) of this project are very similar. This is because tactics like "*simp*?", "*exact*?", and "*apply*?" did not hep much, which is perhaps due to the presence of highly layered and complex definitions (another side effect of the direct approach). Even later, when "[@*simps*]" were added appropriately, the situation did not improve much.

A slight inconvenience was encountered when using the in-built "*Cocone*" structure. Instead of a direct proof of (P, ϕ) being a co-cone of F, "*Cocone*" requires a natural transformation $\tau : F \longrightarrow P_c$, where P_c is the constant functor mapping to P. Although the naturality of τ and the co-cone condition are equivalent, it was slightly cumbersome to figure out this extra step. But perhaps the "*Cocone*" structure is defined like this because it provides a more efficient way of encoding the same information in Lean. All other structures closely corresponded to the mathematical literature.

References

[1] Mac Lane S. *Categories for the Working Mathematician.* 2nd ed. New York: Springer; 1998. (Graduate Texts in Mathematics; vol. 5). p. 76–77.