Emily Riehl

Johns Hopkins University

The ∞-cosmos project

Formalizing 1-, 2-, $\mathcal{V}$-, and ∞-category theory in Lean

Abstract

The ∞-cosmos project aims to leverage the existing libraries developing

- 1-category theory,
- 2-category theory, and
- enriched ($\mathcal{V}$-)category theory

in Lean to formalize basic ∞-category theory.

After giving a high-level overview of the problem, plan, and progress of the project (so far), we illustrate some challenges that we have encountered related to the formalization of supporting results from 1-category theory, 2-category theory, and $\mathcal{V}$-category theory in hopes of attracting interest from folks who want to help us solve them.

Plan

1. What are ∞-categories?

2. The problem, the plan, and progress

3. Challenges

1

What are ∞-categories?

The idea of an ∞-category

Lean defines an ordinary 1-category as follows:

```
class Quiver (V : Type u) where
  /-- The type of edges/arrows/morphisms between a given source and target. -/
  Hom : V → V → Sort v
```

```
class CategoryStruct (obj : Type u) extends Quiver.{v + 1} obj : Type max u (v + 1) where
  /-- The identity morphism on an object. -/
  id : ∀ X : obj, Hom X X
  /-- Composition of morphisms in a category, written `f ≫ g`. -/
  comp : ∀ {X Y Z : obj}, (X ⟶ Y) → (Y ⟶ Z) → (X ⟶ Z)
```

```
class Category (obj : Type u) extends CategoryStruct.{v} obj : Type max u (v + 1) where
  /-- Identity morphisms are left identities for composition. -/
  id_comp : ∀ {X Y : obj} (f : X ⟶ Y), 𝟙 X ≫ f = f := by aesop_cat
  /-- Identity morphisms are right identities for composition. -/
  comp_id : ∀ {X Y : obj} (f : X ⟶ Y), f ≫ 𝟙 Y = f := by aesop_cat
  /-- Composition in a category is associative. -/
  assoc : ∀ {W X Y Z : obj} (f : W ⟶ X) (g : X ⟶ Y) (h : Y ⟶ Z), (f ≫ g) ≫ h = f ≫ g ≫ h := by
    aesop_cat
```

The idea of an ∞-category is just to

- replace all the types by ∞-groupoids aka homotopy types aka anima, i.e., the information of a topological space encoded by its homotopy groups
- and suitably weaken all the structures and axioms.

Quasi-categories in Lean

An elegant "coordinatization" of these ideas encodes an ∞-category as a quasi-category, which Johan Commelin contributed to `Mathlib`:

```
/-- A simplicial set `S` is a *quasicategory* if it satisfies the following horn-filling condition:
for every `n : ℕ` and `0 < i < n`,
every map of simplicial sets `σ₀ : Λ[n, i] → S` can be extended to a map `σ : Δ[n] → S`.

[Kerodon, 003A] -/
class Quasicategory (S : SSet) : Prop where
  hornFilling' : ∀ ⦃n : ℕ⦄ ⦃i : Fin (n+3)⦄ (σ₀ : Λ[n+2, i] ⟶ S)
    (_h0 : 0 < i) (_hn : i < Fin.last (n+2)),
      ∃ σ : Δ[n+2] ⟶ S, σ₀ = hornInclusion (n+2) i ≫ σ
```

Here a simplicial set[1] S is a quasi-category if it satisfies a certain property: namely if any inner horn σ_0 in S can be extended to a simplex σ.

$$\begin{tikzcd} \Lambda[n+2, i] \arrow[r, "\sigma_0"] \arrow[d, hook, "\text{hornInclusion }(n+2)\ i"'] & S \\ \Delta[n+2] \arrow[ur, dashed, "\sigma"'] & \end{tikzcd}$$

[1]A simplicial set is a contravariant functor from a certain SimplexCategory to Type.

How are quasi-categories ∞-categories?

A similar "coordinatization" of the notion of ∞-groupoid is as a Kan complex, a simplicial set in which any outer horn can be extended to a simplex.

```
/-- A simplicial set `S` is a *Kan complex* if it satisfies the following horn-filling condition:
for every `n : ℕ` and `0 ≤ i ≤ n`,
every map of simplicial sets `σ₀ : Λ[n, i] → S` can be extended to a map `σ : Δ[n] → S`. -/
class KanComplex (S : SSet) : Prop where
  hornFilling : ∀ ⦃n : ℕ⦄ ⦃i : Fin (n+1)⦄ (σ₀ : Λ[n, i] ⟶ S),
    ∃ σ : Δ[n] ⟶ S, σ₀ = hornInclusion n i ≫ σ
```

Then

- the maximal sub Kan complex in a quasi-category S defines the ∞-groupoid of objects,
- a certain pullback of the exponential sHom$(\Delta[1], S)$ defines the ∞-groupoid of arrows between two objects,
- n-ary composition can be shown to be well-defined up to a contractible ∞-groupoid of choices.

None of this has been formalized in `Mathlib`.

2

The problem, the plan, and progress

The problem and the plan

The objective is to add the theory of ∞-categories to `Mathlib`, not just the definition. Textbooks that develop that theory using quasi-categories (Lurie's *Higher Topos Theory*, Cisinski's *Higher Categories and Homotopical Algebra*) tend to be very long.

The idea of the ∞-cosmos project is to develop the theory of ∞-categories more abstractly, using the axiomatic notion of an ∞-cosmos, which is an enriched category whose objects are ∞-categories.

From this we can extract a 2-category whose objects are ∞-categories, whose morphisms are ∞-functors, and whose 2-cells are ∞-natural transformations. The formal theory of ∞-categories (adjunctions, co/limits, Kan extensions) can be defined using this 2-category and some of these notions are in the `Mathlib` already!

Proving that quasi-categories define an ∞-cosmos will be hard, but this tedious verifying of homotopy coherences will only need to be done once rather than in every proof.

Progress

The ∞-cosmos project was launched in September 2024. After adding some background material on enriched category theory, we have formalized the following definition:

1.2.1. Definition (∞-cosmos). An **∞-cosmos** $\mathcal{K}$ is a category that is enriched over quasi-categories,[13] meaning in particular that

- its morphisms $f: A \to B$ define the vertices of a quasi-category denoted $\mathsf{Fun}(A, B)$ and referred to as a **functor space**,

that is also equipped with a specified collection of maps that we call **isofibrations** and denote by "$\twoheadrightarrow$" satisfying the following two axioms:

(i) (completeness) The quasi-categorically enriched category $\mathcal{K}$ possesses a terminal object, small products, pullbacks of isofibrations, limits of countable towers of isofibrations, and cotensors with simplicial sets, each of these limit notions satisfying a universal property that is enriched over simplicial sets.[14]

(ii) (isofibrations) The isofibrations contain all isomorphisms and any map whose codomain is the terminal object; are closed under composition, product, pullback, forming inverse limits of towers, and Leibniz cotensors with monomorphisms of simplicial sets; and have the property that if $f: A \twoheadrightarrow B$ is an isofibration and X is any object then $\mathsf{Fun}(X, A) \twoheadrightarrow \mathsf{Fun}(X, B)$ is an isofibration of quasi-categories.

A formalized definition of an ∞-cosmos

```
variable (K : Type u) [Category.{v} K] [SimplicialCategory K]
```

```
/-- A `PreInfinityCosmos` is a simplicially enriched category whose hom-spaces are quasi-categories
and whose morphisms come equipped with a special class of isofibrations.-/
class PreInfinityCosmos extends SimplicialCategory K where
  [has_qcat_homs : ∀ {X Y : K}, SSet.Quasicategory (EnrichedCategory.Hom X Y)]
  IsIsofibration : MorphismProperty K
```

```
variable (K : Type u) [Category.{v} K][PreInfinityCosmos.{v} K]
```

```
/-- An `InfinityCosmos` extends a `PreInfinityCosmos` with limit and isofibration axioms..-/
class InfinityCosmos extends PreInfinityCosmos K where
  comp_isIsofibration {A B C : K} (f : A ↠ B) (g : B ↠ C) : IsIsofibration (f.1 ≫ g.1)
  iso_isIsofibration {X Y : K} (e : X ⟶ Y) [IsIso e] : IsIsofibration e
  all_objects_fibrant {X Y : K} (hY : IsConicalTerminal Y) (f : X ⟶ Y) : IsIsofibration f
  [has_products : HasConicalProducts K]
  prod_map_fibrant {γ : Type w} {A B : γ → K} (f : ∀ i, A i ↠ B i) :
    IsIsofibration (Limits.Pi.map (λ i ↦ (f i).1))
  [has_isoFibration_pullbacks {E B A : K} (p : E ↠ B) (f : A ⟶ B)  : HasConicalPullback p.1 f]
  pullback_is_isoFibration {E B A P : K} (p : E ↠ B) (f : A ⟶ B)
    (fst : P ⟶ E) (snd : P ⟶ A) (h : IsPullback fst snd p.1 f) : IsIsofibration snd
  [has_limits_of_towers (F : ℕᵒᵖ ⥤ K) :
    (∀ n : ℕ, IsIsofibration (F.map (homOfLE (Nat.le_succ n)).op)) → HasConicalLimit F]
  has_limits_of_towers_isIsofibration (F : ℕᵒᵖ ⥤ K) (hf) :
    haveI := has_limits_of_towers F hf
    IsIsofibration (limit.π F (.op 0))
  [has_cotensors : HasCotensors K]
  leibniz_cotensor  {U V : SSet} (i : U ⟶ V) [Mono i] {A B : K} (f : A ↠ B) {P : K}
    (fst : P ⟶ U ⋔ A) (snd : P ⟶ V ⋔ B)
    (h : IsPullback fst snd (cotensorCovMap U f.1) (cotensorContraMap i B)) :
    IsIsofibration (h.isLimit.lift <|
      PullbackCone.mk (cotensorContraMap i A) (cotensorCovMap V f.1)
        (cotensor_bifunctoriality i f.1)) --TODO : Prove that these pullbacks exist.
  local_isoFibration {X A B : K} (f : A ↠ B) : Isofibration (toFunMap X f.1)
```

Related contributions to `Mathlib`

One successful aspect of our project is the rapid rate of contributions to `Mathlib`:

- codiscrete categories (Alvaro Belmonte)
- reflexive quivers (Mario Carneiro, Pietro Monticone, Emily Riehl)
- the opposite category of an enriched category (Daniel Carranza)
- a closed monoidal category is enriched in itself (Daniel Carranza, Joël Riou)
- StrictSegal simplicial sets are 2-coskeletal (Mario Carneiro and Joël Riou)
- StrictSegal simplicial sets are quasicategories (Johan Commelin, Emily Riehl, Nick Ward)
- left and right lifting properties (Jack McKoen)
- SSet.hoFunctor, which constructs a category from a simplicial set (Mario Carneiro, Pietro Monticone, Emily Riehl, Joël Riou)
- SimplicialSet (co)skeleton properties (Mario Carneiro, Pietro Monticone, Emily Riehl, Joël Riou)

A key challenge is the extraordinary demands this has placed on Joël Riou as a reviewer.

3

Challenges

A challenge from 1-category theory

To define the 2-categorical quotient of an ∞-cosmos (WIP), Mario Carneiro and I introduced reflexive quivers

```
/-- A reflexive quiver extends a quiver with a specified arrow `id X : X ⟶ X` for each `X` in its
type of objects. We denote these arrows by `id` since categories can be understood as an extension
of refl quivers.
-/
class ReflQuiver (obj : Type u) extends Quiver.{v} obj : Type max u v where
  /-- The identity morphism on an object. -/
  id : ∀ X : obj, Hom X X
```

and formalized the free category and underlying reflexive quiver adjunction between Cat and ReflQuiv. This is now in `Mathlib`:

```
/--
The adjunction between forming the free category on a reflexive quiver, and forgetting a category
to a reflexive quiver.
-/
nonrec def adj : Cat.freeRefl.{max u v, u} ⊣ ReflQuiv.forget :=
  Adjunction.mkOfUnitCounit {
```

A challenge from 1-category theory, continued

```
left_triangle := by
  ext V
  apply Cat.FreeRefl.lift_unique'
  simp only [id_obj, Cat.free_obj, comp_obj, Cat.freeRefl_obj_α, NatTrans.comp_app,
    forget_obj, whiskerRight_app, associator_hom_app, whiskerLeft_app, id_comp,
    NatTrans.id_app']
  rw [Cat.id_eq_id, Cat.comp_eq_comp]
  simp only [Cat.freeRefl_obj_α, Functor.comp_id]
  rw [← Functor.assoc, ← Cat.freeRefl_naturality, Functor.assoc]
  dsimp [Cat.freeRefl]
  rw [adj.counit.component_eq' (Cat.FreeRefl V)]
  conv =>
    enter [1, 1, 2]
    apply (Quiv.comp_eq_comp (X := Quiv.of _) (Y := Quiv.of _) (Z := Quiv.of _) ..).symm
  rw [Cat.free.map_comp]
  show (_ ⋙ ((Quiv.forget ⋙ Cat.free).map (X := Cat.of _) (Y := Cat.of _)
    (Cat.FreeRefl.quotientFunctor V))) ⋙ _ = _
  rw [Functor.assoc, ← Cat.comp_eq_comp]
  conv => enter [1, 2]; apply Quiv.adj.counit.naturality
  rw [Cat.comp_eq_comp, ← Functor.assoc, ← Cat.comp_eq_comp]
  conv => enter [1, 1]; apply Quiv.adj.left_triangle_components V.toQuiv
  exact Functor.id_comp _
```

Lean was confused by

- what category we're in when objects are type classes
- competing notations for functors
- whiskered commutative diagrams

A challenge from enriched category theory

What is an enriched category?

> To borrow a distinction used by Peter May, the term "enriched" can be used as a compound noun — *enriched categories* — or as an adjective — *enriched* categories. In the noun form, an enriched category $\mathcal{C}$ has no preexisting underlying ordinary category, although we shall see … that the underlying unenriched 1-category can always be identified a posteriori. When used as an adjective, an enriched category $\mathcal{C}$ is perhaps most naturally an ordinary category, whose hom-sets can be given additional structure.
>
> — *Elements of ∞-Category Theory*, Appendix A

`Mathlib` has both notions, referred to as enriched categories and enriched ordinary categories, respectively.

A challenge from enriched category theory, continued

```
variable (V : Type v) [Category.{w} V] [MonoidalCategory V]
class EnrichedCategory (C : Type u₁) where
  Hom : C → C → V
  id (X : C) : 𝟙_ V ⟶ Hom X X
  comp (X Y Z : C) : Hom X Y ⊗ Hom Y Z ⟶ Hom X Z
  id_comp (X Y : C) : (λ_ (Hom X Y)).inv ≫ id X ▷ _ ≫ comp X X Y = 𝟙 _ := by aesop_cat
  comp_id (X Y : C) : (ρ_ (Hom X Y)).inv ≫ _ ◁ id Y ≫ comp X Y Y = 𝟙 _ := by aesop_cat
  assoc (W X Y Z : C) : (α_ _ _ _).inv ≫ comp W X Y ▷ _ ≫ comp W Y Z =
    _ ◁ comp X Y Z ≫ comp W X Z := by aesop_cat
```

```
variable (V : Type u') [Category.{v'} V] [MonoidalCategory V]
  (C : Type u) [Category.{v} C]
/-- An enriched ordinary category is a category `C` that is also enriched
over a category `V` in such a way that morphisms `X ⟶ Y` in `C` identify
to morphisms `𝟙_ V ⟶ (X ⟶[V] Y)` in `V`. -/
class EnrichedOrdinaryCategory extends EnrichedCategory V C where
  /-- morphisms `X ⟶ Y` in the category identify morphisms
    `𝟙_ V ⟶ (X ⟶[V] Y)` in `V` -/
  homEquiv {X Y : C} : (X ⟶ Y) ≃ (𝟙_ V ⟶ (X ⟶[V] Y))
  homEquiv_id (X : C) : homEquiv (𝟙 X) = eId V X := by aesop_cat
  homEquiv_comp {X Y Z : C} (f : X ⟶ Y) (g : Y ⟶ Z) :
    homEquiv (f ≫ g) = (λ_ _).inv ≫ (homEquiv f ⊗ homEquiv g) ≫
      eComp V X Y Z := by aesop_cat
```

The enriched categories literature is less clear about this distinction.

A challenge from 2-category theory

On paper, 2-cells in a 2-category compose by pasting:

$$\begin{tikzcd}
& A \arrow[r, "G_1"] \arrow[d, "L_1" description] \arrow[dr, phantom, "\Swarrow\alpha"] & C \arrow[r, equal] \arrow[d, "L_2" description] \arrow[r, phantom, "\Swarrow\eta_2"', near start] & C \arrow[r, "G_2"] \arrow[d, "L_2" description] \arrow[dr, phantom, "\Swarrow\beta"] & E \arrow[r, equal] \arrow[d, "L_3" description] \arrow[r, phantom, "\Swarrow\eta_3"', near start] & E \\
B \arrow[r, equal] \arrow[ur, "R_1"] \arrow[r, phantom, "\Swarrow\epsilon_1", near end] & B \arrow[r, "H_1"'] & D \arrow[r, equal] \arrow[ur, "R_2" description] \arrow[r, phantom, "\Swarrow\epsilon_2", near end] & D \arrow[r, "H_2"'] & F \arrow[ur, "R_3"'] &
\end{tikzcd}$$

In `Mathlib`, the 2-cells displayed here belong to dependent types (over their boundary 1-cells and objects) and depending on how the whiskerings are encoded are not obviously composable at all:

e.g., is $R_3H_2L_2\eta_2G_1R_1$ composable with $R_3H_2\epsilon_2L_2G_1R_1$?

A challenge from 2-category theory

```
/-- The mates equivalence commutes with vertical composition. -/
theorem mateEquiv_vcomp
    (α : G₁ ⋙ L₂ ⟶ L₁ ⋙ H₁) (β : G₂ ⋙ L₃ ⟶ L₂ ⋙ H₂) :
    (mateEquiv (G := G₁ ⋙ G₂) (H := H₁ ⋙ H₂) adj₁ adj₃) (leftAdjointSquare.vcomp α β) =
      rightAdjointSquare.vcomp (mateEquiv adj₁ adj₂ α) (mateEquiv adj₂ adj₃ β) := by
  unfold leftAdjointSquare.vcomp rightAdjointSquare.vcomp mateEquiv
  ext b
  simp only [comp_obj, Equiv.coe_fn_mk, whiskerLeft_comp, whiskerLeft_twice, whiskerRight_comp,
    assoc, comp_app, whiskerLeft_app, whiskerRight_app, id_obj, Functor.comp_map,
    whiskerRight_twice]
  slice_rhs 1 4 => rw [← assoc, ← assoc, ← unit_naturality (adj₃)]
  rw [L₃.map_comp, R₃.map_comp]
  slice_rhs 2 4 =>
    rw [← R₃.map_comp, ← R₃.map_comp, ← assoc, ← L₃.map_comp, ← G₂.map_comp, ← G₂.map_comp]
    rw [← Functor.comp_map G₂ L₃, β.naturality]
  rw [(L₂ ⋙ H₂).map_comp, R₃.map_comp, R₃.map_comp]
  slice_rhs 4 5 =>
    rw [← R₃.map_comp, Functor.comp_map L₂ _, ← Functor.comp_map _ L₂, ← H₂.map_comp]
    rw [adj₂.counit.naturality]
  simp only [comp_obj, Functor.comp_map, map_comp, id_obj, Functor.id_map, assoc]
  slice_rhs 4 5 =>
    rw [← R₃.map_comp, ← H₂.map_comp, ← Functor.comp_map _ L₂, adj₂.counit.naturality]
  simp only [comp_obj, id_obj, Functor.id_map, map_comp, assoc]
  slice_rhs 3 4 =>
    rw [← R₃.map_comp, ← H₂.map_comp, left_triangle_components]
  simp only [map_id, id_comp]
```

In the 2-category Cat, I formalized a proof that the unit η_2 and counit ϵ_2 cancel, but not via a 2-categorical pasting argument. As a result, `Mathlib` does not know this is true in any 2-category.

Contributors to the ∞-cosmos project

Pietro Monticone and Patrick Massot helped us set up a blueprint (and website) to organize the workflow. So far formalizations (and preliminary mathematical work) have been contributed by:

Dagur Asgeirsson, Alvaro Belmonte, Mario Carneiro, Daniel Carranza, Johan Commelin, Jack McKoen, Pietro Monticone, Matej Penciak, Nima Rasekh, Emily Riehl, Joël Riou, Joseph Tooby-Smith, Adam Topaz, Dominic Verity, Nick Ward, and Zeyi Zhao.

Anyone is welcome to join us!

emilyriehl.github.io/infinity-cosmos

Thank you!