



Mario Carneiro and Emily Riehl

Chalmers University of Technology  
Johns Hopkins University

## Formalizing colimits in $\mathcal{C}at$

ITP 2025

## An unlikely partnership



This project came out of the semester on formal mathematics at the Hausdorff Institute of Mathematics in summer 2024.

- I am a Lean formalization expert who took a category theory class once.
- Emily is an  $\infty$ -category theorist with a bit of Lean experience.

So I suggested that we try to define the fundamentals of  $\infty$ -category theory in Lean.

As it turns out,  $\infty$ -categories are complicated to define, but you can abstract the details and define the rules that a reasonable definition of  $\infty$ -categories should satisfy; this is called an  $\infty$ -*cosmos* and the  $\infty$ -cosmos project came out of this.

# The $\infty$ -cosmos project: overview



The  $\infty$ -cosmos project aims to use a convenient abstraction boundary to formalize some core category theory of  $\infty$ -categories for immediate use in Mathlib.

# The $\infty$ -cosmos project: overview

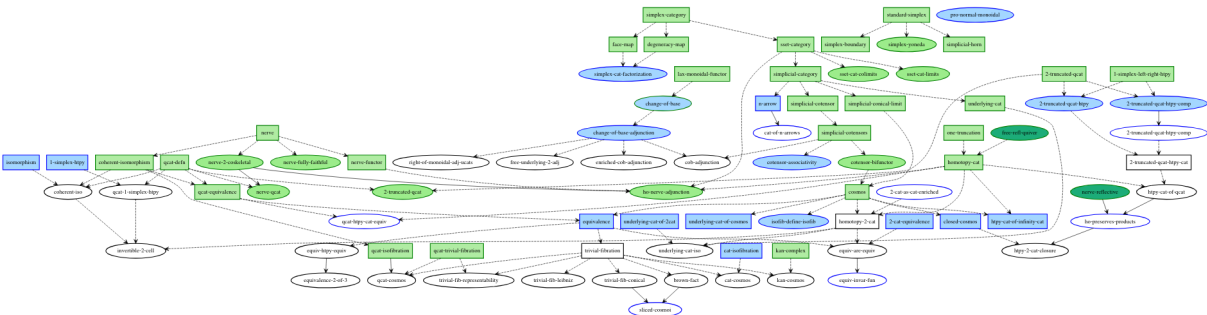


The  $\infty$ -cosmos project aims to use a convenient abstraction boundary to formalize some core category theory of  $\infty$ -categories for immediate use in Mathlib.

Mathlib knows definitions of

- $\infty$ -groupoid, that is a weak infinite-dimensional category with all morphisms invertible, modeled as [Kan complexes](#)
- $\infty$ -category, that is a weak infinite-dimensional category with non-invertible morphisms only allowed in dimension one, modeled as [quasi-categories](#)

But very few theorems have been formalized about these notions. In particular, earlier this year, Joël Riou noticed that the definition of Kan complexes was wrong!



## From $\infty$ -categories to 2-categories



There is a 2-category (aka strict bicategory) whose objects are  $\infty$ -categories, whose morphisms are  $\infty$ -functors, and whose 2-cells are  $\infty$ -natural transformations. This simplifies matters because we can make use of the existing work on 2-categories in Mathlib.

Key to the construction is the **nerve of a category**, and its left adjoint the **homotopy functor**.

# From $\infty$ -categories to 2-categories



There is a 2-category (aka strict bicategory) whose objects are  $\infty$ -categories, whose morphisms are  $\infty$ -functors, and whose 2-cells are  $\infty$ -natural transformations. This simplifies matters because we can make use of the existing work on 2-categories in Mathlib.

Key to the construction is the **nerve of a category**, and its left adjoint the **homotopy functor**.

Mathlib already had the definition of **nerve**; this talk will be about how we constructed the **homotopy functor**.

# Simplicial sets



The **simplex category**,  $\Delta$ , is the category which has an object for each natural number denoted  $[n]$  thought of as  $\text{Fin}(n + 1)$ , and whose morphisms  $[n] \rightarrow [m]$  are monotone maps  $\text{Fin}(n + 1) \rightarrow \text{Fin}(m + 1)$ .

For example, there is one morphism  $i : [1] \rightarrow [0]$  which sends both 0 and 1 to 0, and two morphisms  $s, t : [0] \rightarrow [1]$  defined by  $s(0) = 0$  and  $t(0) = 1$ .



# Simplicial sets



The **simplex category**,  $\Delta$ , is the category which has an object for each natural number denoted  $[n]$  thought of as  $\text{Fin}(n + 1)$ , and whose morphisms  $[n] \rightarrow [m]$  are monotone maps  $\text{Fin}(n + 1) \rightarrow \text{Fin}(m + 1)$ .

For example, there is one morphism  $i : [1] \rightarrow [0]$  which sends both 0 and 1 to 0, and two morphisms  $s, t : [0] \rightarrow [1]$  defined by  $s(0) = 0$  and  $t(0) = 1$ .

A **simplicial set** is a functor  $X : \Delta^{\text{op}} \Rightarrow \text{Type}$ .

So a simplicial set has sets  $X[0]$ ,  $X[1]$ ,  $X[2]$ , etc. which we interpret as “objects”, “morphisms” and in general “ $n$ -cells”, and the functions  $X(s), X(t) : X[1] \rightarrow X[0]$  pick out the source and target of a morphism / 1-cell, while  $X(i) : X[0] \rightarrow X[1]$  gives the identity morphism on an object.

# Simplicial sets



The **simplex category**,  $\Delta$ , is the category which has an object for each natural number denoted  $[n]$  thought of as  $\text{Fin}(n + 1)$ , and whose morphisms  $[n] \rightarrow [m]$  are monotone maps  $\text{Fin}(n + 1) \rightarrow \text{Fin}(m + 1)$ .

For example, there is one morphism  $i : [1] \rightarrow [0]$  which sends both 0 and 1 to 0, and two morphisms  $s, t : [0] \rightarrow [1]$  defined by  $s(0) = 0$  and  $t(0) = 1$ .

A **simplicial set** is a functor  $X : \Delta^{\text{op}} \Rightarrow \text{Type}$ .

So a simplicial set has sets  $X[0]$ ,  $X[1]$ ,  $X[2]$ , etc. which we interpret as “objects”, “morphisms” and in general “ $n$ -cells”, and the functions  $X(s), X(t) : X[1] \rightarrow X[0]$  pick out the source and target of a morphism / 1-cell, while  $X(i) : X[0] \rightarrow X[1]$  gives the identity morphism on an object.

So even with these rudimentary definitions we can see something like an  $\infty$ -category structure taking shape.

# The nerve of a category



Given a category  $\mathcal{C}$ ,  $\text{nerve } \mathcal{C}$  is a simplicial set, whose type of  $n$ -cells  $(\text{nerve } \mathcal{C})[n]$  is the collection of functors  $[n] \rightarrow \mathcal{C}$ , where  $[n]$  is the preorder category:

$$0 \longrightarrow 1 \longrightarrow \cdots \longrightarrow n-1 \longrightarrow n$$

# The nerve of a category



Given a category  $\mathcal{C}$ ,  $\text{nerve } \mathcal{C}$  is a simplicial set, whose type of  $n$ -cells  $(\text{nerve } \mathcal{C})[n]$  is the collection of functors  $[n] \rightarrow \mathcal{C}$ , where  $[n]$  is the preorder category:

$$0 \longrightarrow 1 \longrightarrow \dots \longrightarrow n-1 \longrightarrow n$$

This assembles into a functor  $\text{nerve} : \mathcal{Cat} \Rightarrow \mathbf{sSet}$  between the category of categories and the category of simplicial sets.

# The nerve of a category



Given a category  $\mathcal{C}$ ,  $\text{nerve } \mathcal{C}$  is a simplicial set, whose type of  $n$ -cells  $(\text{nerve } \mathcal{C})[n]$  is the collection of functors  $[n] \rightarrow \mathcal{C}$ , where  $[n]$  is the preorder category:

$$0 \longrightarrow 1 \longrightarrow \dots \longrightarrow n-1 \longrightarrow n$$

This assembles into a functor  $\text{nerve} : \mathbf{Cat} \Rightarrow \mathbf{sSet}$  between the category of categories and the category of simplicial sets.

Our objective is to define the left adjoint of this functor, which takes a simplicial set to its *homotopy category*.

# The homotopy category



The action of  $\mathbf{ho} : \mathbf{sSet} \Rightarrow \mathbf{Cat}$  is to take a simplicial set  $X$  to its quotient 1-category  $\mathbf{ho} X$ . Intuitively, the objects will be  $X[0]$  and the morphisms will be formal composites of arrows from  $X[1]$ , except that two morphisms are identified whenever there is a 2-cell

$$\begin{array}{ccc} & y & \\ f \nearrow & \sigma & \searrow g \\ x & \xrightarrow{h} & z \end{array} \quad \text{witnessing that } g \circ f = h.$$

# The homotopy category



The action of  $\mathbf{ho} : \mathbf{sSet} \Rightarrow \mathbf{Cat}$  is to take a simplicial set  $X$  to its quotient 1-category  $\mathbf{ho} X$ . Intuitively, the objects will be  $X[0]$  and the morphisms will be formal composites of arrows from  $X[1]$ , except that two morphisms are identified whenever there is a 2-cell

$$\begin{array}{ccc} & y & \\ f \nearrow & \sigma & \searrow g \\ x & \xrightarrow{h} & z \end{array} \quad \text{witnessing that } g \circ f = h.$$

## Theorem

The nerve functor admits a left adjoint defined by the functor that sends a simplicial set to its homotopy category:  $\mathbf{Cat} \xrightleftharpoons[\text{nerve}]{\mathbf{ho}} \mathbf{Set}^{\Delta^{op}}$ . The nerve functor is full and faithful.

## An abstract nonsense proof



This adjunction exists as a special case of a result already in Mathlib by Mehta and Riou:

### Theorem

Suppose  $A: \mathcal{A} \rightarrow \mathcal{E}$  is any functor, where  $\mathcal{A}$  is small and  $\mathcal{E}$  is cocomplete. Then the left Kan extension of  $A$  along the Yoneda embedding  $\mathcal{Y}: \mathcal{A} \rightarrow \mathbf{Set}^{\mathcal{A}^{op}}$  is left adjoint to the

restricted Yoneda functor:  $\mathcal{E} \xrightleftharpoons[\mathcal{E}(A-, -)]{\mathrm{lan}_{\mathcal{Y}} A} \mathbf{Set}^{\mathcal{A}^{op}}$ .

The nerve adjunction is an example of a Yoneda adjunction defined relative to the functor `SimplexCategory.toCat` :  $\Delta \Rightarrow \mathbf{Cat}$  which sends  $[n] : \Delta$  to  $[n] : \mathbf{Cat}$ .



# An abstract nonsense proof



This adjunction exists as a special case of a result already in Mathlib by Mehta and Riou:

## Theorem

Suppose  $A: \mathcal{A} \rightarrow \mathcal{E}$  is any functor, where  $\mathcal{A}$  is small and  $\mathcal{E}$  is cocomplete. Then the left Kan extension of  $A$  along the Yoneda embedding  $\mathcal{Y}: \mathcal{A} \rightarrow \mathbf{Set}^{\mathcal{A}^{op}}$  is left adjoint to the

restricted Yoneda functor:  $\mathcal{E} \xrightleftharpoons[\mathcal{E}(A-, -)]{\mathrm{lan}_{\mathcal{Y}} A} \mathbf{Set}^{\mathcal{A}^{op}}.$

The nerve adjunction is an example of a Yoneda adjunction defined relative to the functor `SimplexCategory.toCat` :  $\Delta \Rightarrow \mathbf{Cat}$  which sends  $[n] : \Delta$  to  $[n] : \mathbf{Cat}$ .

Unfortunately, applying this requires that  $\mathcal{E} := \mathbf{Cat}$  is cocomplete, i.e.  $\mathbf{Cat}$  has colimits.

## A circular proof



Consulting the literature, we are reminded that colimits in  $\mathcal{C}at$  are constructed as a corollary of the nerve adjunction we wanted in the first place: since the nerve functor is fully faithful, the adjunction exhibits  $\mathcal{C}at$  as a reflective subcategory of the cocomplete category  $sSet := Set^{\Delta^{op}}$ , from which the result follows by a theorem formalized by Morrison, McKoen and Mehta:

```
-- If `C` has colimits then any reflective subcategory has colimits. -/
theorem hasColimits_of_reflective (R : D  $\Rightarrow$  C)
  [Reflective R] [HasColimitsOfSize.{v, u} C] :
  HasColimitsOfSize.{v, u} D :=
  ⟨fun _  $\Rightarrow$  hasColimitsOfShape_of_reflective R⟩
```

## A circular proof



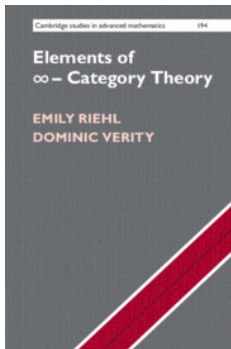
Consulting the literature, we are reminded that colimits in  $\mathcal{C}at$  are constructed as a corollary of the nerve adjunction we wanted in the first place: since the nerve functor is fully faithful, the adjunction exhibits  $\mathcal{C}at$  as a reflective subcategory of the cocomplete category  $sSet := Set^{\Delta^{op}}$ , from which the result follows by a theorem formalized by Morrison, McKoen and Mehta:

```
-- If `C` has colimits then any reflective subcategory has colimits. -/
theorem hasColimits_of_reflective (R : D  $\Rightarrow$  C)
  [Reflective R] [HasColimitsOfSize.{v, u} C] :
  HasColimitsOfSize.{v, u} D :=
  {fun _  $\Rightarrow$  hasColimitsOfShape_of_reflective R}
```

So we scrapped the abstract nonsense proof and set out to construct the adjunction directly.

# A pen-and-paper proof

A joint book Emily had written with **Dominic Verity** reviews the construction of the reflective embedding of 1-categories into  $\infty$ -categories in less than one page:



1.1.10. DEFINITION (the homotopy category [44, §2.4]). By 1-truncating, any simplicial set  $X$  has an underlying reflexive directed graph with the 0-simplices of  $X$  defining the objects and the 1-simplices defining the arrows:

$$X_1 \begin{array}{c} \xrightarrow{\delta^1} \\ \xleftarrow{\delta^0} \end{array} X_0,$$

By convention, the source of an arrow  $f \in X_1$  is its 0th face  $f \cdot \delta^1$  (the face opposite 1) while the target is its 1st face  $f \cdot \delta^0$  (the face opposite 0). The **free category** on this reflexive directed graph has  $X_0$  as its object set, degenerate 1-simplices serving as identity morphisms, and nonidentity morphisms defined to be finite directed paths of nondegenerate 1-simplices. The **homotopy category**  $\mathbf{h}X$  of  $X$  is the quotient of the free category on its underlying reflexive directed graph by the congruence<sup>3</sup> generated by imposing a composition relation  $h = g \circ f$  witnessed by 2-simplices

$$\begin{array}{ccc} & x_1 & \\ f \nearrow & & \searrow g \\ x_0 & \xrightarrow{h} & x_2 \end{array}$$

This relation implies in particular that homotopic 1-simplices represent the same arrow in the homotopy category.

The homotopy category of the nerve of a 1-category is isomorphic to the original category, as the 2-simplices in the nerve witness all of the composition relations satisfied by the arrows in the underlying reflexive directed graph. Indeed, the natural isomorphism  $\mathbf{h}C \cong C$  forms the counit of an adjunction, embedding  $\mathbf{Cat}$  as a reflective subcategory of  $\mathbf{sSet}$ .

1.1.11. PROPOSITION. *The nerve embedding admits a left adjoint, namely the functor which sends a simplicial set to its homotopy category:*

$$\mathbf{Cat} \begin{array}{c} \xleftarrow{h} \\ \perp \\ \xrightarrow{\quad} \end{array} \mathbf{sSet}$$

The adjunction of Proposition 1.1.11 exists for formal reasons (see Exercise 1.1.i), but nevertheless, a direct proof can be enlightening.

PROOF. For any simplicial set  $X$ , there is a natural map from  $X$  to the nerve of its homotopy category  $\mathbf{h}X$ ; since nerves are 2-coskeletal, it suffices to define the map  $\mathrm{sk}_2 X \rightarrow \mathbf{h}X$ , and this is given immediately by the construction of Definition 1.1.10. Note that the quotient map  $X \rightarrow \mathbf{h}X$  becomes an isomorphism upon applying the homotopy category functor and is already an isomorphism whenever  $X$  is the nerve of a category. Thus the adjointness follows from Lemma B.4.2 or by direct verification of the triangle equalities.  $\square$

# A formalized proof





It took the two of us three months (part time) to formalize this result in **Lean**.


It then took another six months for this code, which totalled 2240 lines split across seven PRs to pass the review process to be integrated into **Lean's Mathlib**.

The paper goes into more detail on the formalization and the challenges we encountered along the way. Needless to say it's a lot more than 1 page.

## Formalizing colimits in $\mathcal{Cat}$

Mario Carneiro  

Chalmers University of Technology, Sweden

Emily Riehl<sup>1</sup>  

Department of Mathematics, Johns Hopkins University, 3400 N Charles Street, Baltimore, MD, USA

---

### Abstract

Certain results involving “higher structures” are not currently accessible to computer formalization because the prerequisite  $\infty$ -category theory has not been formalized. To support future work on formalizing  $\infty$ -category theory in Lean’s mathematics library, we formalize some fundamental constructions involving the 1-category of categories. Specifically, we construct the left adjoint to the nerve embedding of categories into simplicial sets, defining the homotopy category functor. We prove further that this adjunction is reflective, which allows us to conclude that  $\mathcal{Cat}$  has colimits. To our knowledge this is the first formalized proof that the category of categories is cocomplete.

# A formalized proof



It took the two of us three months (part time) to formalize this result in **Lean**.

It then took another six months for this code, which totalled 2240 lines split across seven PRs to pass the review process to be integrated into **Lean's Mathlib**.

The paper goes into more detail on the formalization and the challenges we encountered along the way. Needless to say it's a lot more than 1 page.

## Formalizing colimits in $\mathcal{Cat}$

Mario Carneiro ✉

Chalmers University of Technology, Sweden

Emily Riehl<sup>1</sup> ✉

Department of Mathematics, Johns Hopkins University, 3400 N Charles Street, Baltimore, MD, USA

---

### Abstract

Certain results involving “higher structures” are not currently accessible to computer formalization because the prerequisite  $\infty$ -category theory has not been formalized. To support future work on formalizing  $\infty$ -category theory in Lean's mathematics library, we formalize some fundamental constructions involving the 1-category of categories. Specifically, we construct the left adjoint to the nerve embedding of categories into simplicial sets, defining the homotopy category functor. We prove further that this adjunction is reflective, which allows us to conclude that  $\mathcal{Cat}$  has colimits. To our knowledge this is the first formalized proof that the category of categories is cocomplete.

What happened here?

# Plan



1. Overview of the proof
2. Difficulties and complications



1

## Overview of the proof

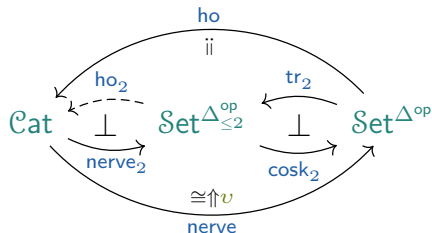


## Decomposing the adjunction



We will build the adjunction as a composition of two other adjunctions, by proving that  $\text{nerve}$  is isomorphic to the composition  $\text{cosk}_2 \circ \text{nerve}_2$  going through the category  $\text{Set}^{\Delta_{\leq 2}^{\text{op}}}$  of **2-truncated simplicial sets**. Given  $n : \mathbb{N}$ , we define the  $n$ -truncated simplex category  $\Delta_{\leq n}$  as the full subcategory  $\Delta_{\leq n} \subset \Delta$  spanned by the objects  $[0], \dots, [n]$ .

$\text{tr}_2 : \text{Set}^{\Delta^{\text{op}}} \Rightarrow \text{Set}^{\Delta_{\leq 2}^{\text{op}}}$  is *2-truncation* (forgetting higher components of a simplicial set).  $\text{cosk}_2$  is its right Kan extension.  $\text{nerve}_2 := \text{tr}_2 \circ \text{nerve}$  just returns the 0,1,2-cells of  $\text{nerve}$  and the adjunction  $\text{tr}_2 \dashv \text{cosk}_2$  allows us to construct  $v$ .



# Decomposing the adjunction



We are left with essentially three subgoals:

- To construct the map  $\mathbf{ho}_2 : \mathbf{Set}^{\Delta_{\leq 2}^{\text{op}}} \rightarrow \mathbf{Cat}$ .
- To prove  $\mathbf{ho}_2 \dashv \mathbf{nerve}_2$ .
- To prove that the nerve is *2-coskeletal*, which is equivalent to the assertion that the natural transformation  $\nu$  is an isomorphism.

# Reflexive quivers



A *quiver* is a “category without identity and composition”: it has a set of objects and morphisms between pairs of objects, and no laws on top. There is an obvious forgetful functor from categories to quivers, with a left adjoint  $\text{Cat} \begin{matrix} \xleftarrow{F} \\ \perp \\ \xrightarrow{U} \end{matrix} \text{Quiv}$  which builds the “free category” generated from given arrows.

For our project we defined *reflexive quivers*, a “category without composition”, adding only the identity morphisms. This decomposes the above adjunction to

$$\text{Cat} \begin{matrix} \xleftarrow{F} \\ \perp \\ \xrightarrow{U} \end{matrix} \text{rQuiv} \begin{matrix} \xleftarrow{F} \\ \perp \\ \xrightarrow{U} \end{matrix} \text{Quiv} \quad \text{giving us the free category from a reflexive quiver.}$$

# The homotopy relation



Recall that we want to construct  $\mathbf{ho}_2 : \mathbf{Set}^{\Delta_{\leq 2}^{\text{op}}} \rightarrow \mathbf{Cat}$ , so consider a 2-truncated simplicial set  $X : \Delta_{\leq 2}^{\text{op}} \Rightarrow \mathbf{Type}$ . First, we use the set  $X[0]$  for the objects, and

$$\{f : X[1] \mid X(s)(f) = A \wedge X(t)(f) = B\}$$

defines  $\text{Hom}(A, B)$ . This is not a category, but it is a reflexive quiver because we have  $X(i)(A) : \text{Hom}(A, A)$ .

We then construct the free category generated by this refl.quiver, whose morphisms are formal composites of the above. There is a general construction for taking the morphism quotient generated from a relation, and we use the relation generated by  $X(\sigma_{01})(\varphi) \circ X(\sigma_{12})(\varphi) \sim X(\sigma_{02})(\varphi)$  for each  $\varphi : X[2]$ , where  $\sigma_{01}, \sigma_{12}, \sigma_{02}$  are the three elements of  $[1] \rightarrow [2]$ . This quotient is (the object part of)  $\mathbf{ho}_2(X) : \mathbf{Cat}$ .

# Constructing the adjunction



The tricky part of the adjunction  $\mathbf{ho}_2 \dashv \mathbf{nerve}_2$  is the unit, which is constructed using the following lemma:

## Theorem

Given  $X : \Delta_{\leq 2}^{op} \Rightarrow \mathbf{Type}$  a 2-truncated simplicial set and  $\mathcal{C}$  a category, consider a reflexive prefunctor  $F : U(X) \rightarrow U(\mathcal{C})$  a map from the r.q. of  $X$  to the r.q. of  $\mathcal{C}$ . If for each  $\varphi : X[2]$ ,  $F(\sigma_{02}(\varphi)) = F(\sigma_{01}(\varphi)) \circ F(\sigma_{01}(\varphi))$  (using composition from  $\mathcal{C}$ ), then  $F$  lifts uniquely to a map  $\hat{F} : X \rightarrow U_2(\mathcal{C})$  in  $\mathbf{sSet}_{\leq 2}$ .

Constructing the lift  $\hat{F}$ , a natural transformation, involves constructing three components for  $\hat{F}[0], \dots, \hat{F}[2]$ ; and because of the definition by cases we have 31 different naturalities to prove (one for each morphism in  $\Delta_{\leq 2}$ ).



2

Difficulties and complications

# Confusions of notation/encoding



Category theory famously eats itself:

- A **category**  $\mathcal{C}$  consists of objects  $A, B, C$  and arrows  $f: A \rightarrow B, g: B \rightarrow C$  with composition and identities satisfying axioms.
- **Categories** themselves assemble into a category  $\mathbf{Cat}$  whose objects are **categories** and whose morphisms are **functors**.

## Confusions of notation/encoding



Category theory famously eats itself:

- A **category**  $\mathcal{C}$  consists of objects  $A, B, C$  and arrows  $f: A \rightarrow B, g: B \rightarrow C$  with composition and identities satisfying axioms.
- **Categories** themselves assemble into a category  $\mathbf{Cat}$  whose objects are **categories** and whose morphisms are **functors**.

In Mathlib, categories are typically unbundled, as types with a category structure, while objects of  $\mathbf{Cat}$  are bundled categories. Functors come with

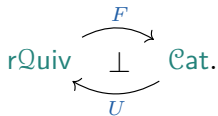
- two different notations whether they are thought of as functors between categories or arrows in  $\mathbf{Cat}$  (“ $\Rightarrow$ ” vs “ $\rightarrow$ ”) and
- distinct notations for composition (“ $\ggg$ ” vs “ $\gg$ ”) and identities (“ $1$ ” vs “ $\mathbb{1}$ ”) in each setting.



# Confusions of notation/encoding



We formalized the **free category** and **underlying reflexive quiver** adjunction



But the construction of the adjunction was repeatedly hindered by confusions about the encoding of categorical data.

```
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 _
```



On paper size issues in category theory are typically addressed in a hand wavy fashion:<sup>1</sup>

REMARK 1.1.5. Russell's paradox implies that there is no set whose elements are "all sets." This is the reason why we have used the vague word "collection" in Definition 1.1.1. Indeed, in each of the examples listed in 1.1.3, the collection of objects is not a set. Eilenberg and Mac Lane address this potential area of concern as follows:

... the whole concept of a category is essentially an auxiliary one; our basic concepts are essentially those of a *functor* and of a natural transformation .... The idea of a category is required only by the precept that every function should have a definite class as domain and a definite class as range, for the categories are provided as the domains and ranges of functors. Thus one could drop the category concept altogether and adopt an even more intuitive standpoint, in which a functor such as "Hom" is not defined over the category of "all" groups, but for each particular pair of groups which may be given. [EM45]

The set-theoretical issues that confront us while defining the notion of a category will compound as we develop category theory further. For that reason, common practice among category theorists is to work in an extension of the usual Zermelo–Fraenkel axioms of set theory, with new axioms allowing one to distinguish between "small" and "large" sets, or between sets and classes. The search for the most useful set-theoretical foundations for category theory is a fascinating topic that unfortunately would require too long of a digression to explore.<sup>12</sup> Instead, we sweep these foundational issues under the rug, not because these issues are not serious or interesting, but because they distract from the task at hand.<sup>13</sup>

---

<sup>1</sup>Footnote 13 begins "If pressed, let us assume that there exists a countable sequence of inaccessible cardinals, ..."

## Universe levels (Emily's take)



Of course this doesn't work in proof assistant and over a year's worth of attempts to formalize some category theory in Lean I find myself drawn between two poles:

- On the one hand, I'm embarrassed by how poorly I understand the implications of various categorical constructions on universe levels.
- On the other, I am frustrated by how often universe errors “distract from the task at hand” and feel increasingly drawn to the dark side of **type-in-type**.

## Universe levels (Emily's take)



Of course this doesn't work in proof assistant and over a year's worth of attempts to formalize some category theory in Lean I find myself drawn between two poles:

- On the one hand, I'm embarrassed by how poorly I understand the implications of various categorical constructions on universe levels.
- On the other, I am frustrated by how often universe errors “distract from the task at hand” and feel increasingly drawn to the dark side of **type-in-type**.

There are two open questions related to practical treatment of universes that I would like to answer:

- What are the best strategies for dealing with universe errors that arise during formalization, both with the aim of resolving them and with the aim of not letting them get in the way of other progress?
- How should universes be addressed in the pen-and-paper literature to better prepare category theorists for formalization, i.e., how should I address this in *Category Theory in Context* volume II?

## Universe levels (Mario's take)



As someone who works on ITP design to some extent, I find it interesting how people like Emily experience universes in practice.

- A common piece of advice I would give is to just ignore universe parametricity and set everything to use Type 0.

## Universe levels (Mario's take)



As someone who works on ITP design to some extent, I find it interesting how people like Emily experience universes in practice.

- A common piece of advice I would give is to just ignore universe parametricity and set everything to use Type 0.
- This would allow us to split the work between doing “real mathematics” and administrative stuff for the universes (which I felt much more competent to do without understanding the surrounding structure)

## Universe levels (Mario's take)



As someone who works on ITP design to some extent, I find it interesting how people like Emily experience universes in practice.

- A common piece of advice I would give is to just ignore universe parametricity and set everything to use Type 0.
- This would allow us to split the work between doing “real mathematics” and administrative stuff for the universes (which I felt much more competent to do without understanding the surrounding structure)
- I think it would actually be a good idea to have a type-in-type / ignore-universes flag for being able to do this systematically.

# Invisible inclusions



A **simplicial set** is a functor  $X: \Delta^{\text{op}} \Rightarrow \text{Type}$ , indexed by a category  $\Delta$  whose objects are natural numbers.

Some constructions require only the data of a **2-truncated simplicial set**, indexed by the subcategory  $\Delta_{\leq 2} \subset \Delta$  spanned by the objects  $[0]$ ,  $[1]$ ,  $[2]$ .



# Invisible inclusions



A **simplicial set** is a functor  $X: \Delta^{\text{op}} \Rightarrow \text{Type}$ , indexed by a category  $\Delta$  whose objects are natural numbers.

Some constructions require only the data of a **2-truncated simplicial set**, indexed by the subcategory  $\Delta_{\leq 2} \subset \Delta$  spanned by the objects  $[0]$ ,  $[1]$ ,  $[2]$ .

On paper,  $\Delta_{\leq 2}$  has just three objects — the natural numbers  $[0]$ ,  $[1]$ , and  $[2]$  — and then has hom-types inherited from  $\Delta$ .

# Invisible inclusions



A **simplicial set** is a functor  $X: \Delta^{\text{op}} \Rightarrow \text{Type}$ , indexed by a category  $\Delta$  whose objects are natural numbers.

Some constructions require only the data of a **2-truncated simplicial set**, indexed by the subcategory  $\Delta_{\leq 2} \subset \Delta$  spanned by the objects  $[0]$ ,  $[1]$ ,  $[2]$ .

On paper,  $\Delta_{\leq 2}$  has just three objects — the natural numbers  $[0]$ ,  $[1]$ , and  $[2]$  — and then has hom-types inherited from  $\Delta$ .

In Mathlib, objects of  $\Delta_{\leq 2}$  are natural numbers  $m$  together with a proof that  $m \leq 2$ .

# Invisible inclusions



A **simplicial set** is a functor  $X: \Delta^{\text{op}} \Rightarrow \text{Type}$ , indexed by a category  $\Delta$  whose objects are natural numbers.

Some constructions require only the data of a **2-truncated simplicial set**, indexed by the subcategory  $\Delta_{\leq 2} \subset \Delta$  spanned by the objects  $[0]$ ,  $[1]$ ,  $[2]$ .

On paper,  $\Delta_{\leq 2}$  has just three objects — the natural numbers  $[0]$ ,  $[1]$ , and  $[2]$  — and then has hom-types inherited from  $\Delta$ .

In Mathlib, objects of  $\Delta_{\leq 2}$  are natural numbers  $m$  together with a proof that  $m \leq 2$ .

This causes all sorts of problems.

# Invisible inclusions



Proofs involving  $\Delta_{\leq 2}$  often require rewriting along an equality between arrows that has been proven for  $\Delta$ . Since `rw` typically fails, we are forced to duplicate infrastructure:

```
-- Abbreviations for face maps in the 2-truncated simplex category. --
abbrev  $\delta_2$  {n} (i : Fin (n + 2)) (hn := by decide) (hn' := by decide) :
  ([n], hn) : SimplexCategory.Truncated 2) → ([n + 1], hn') := SimplexCategory. $\delta$  i

-- Abbreviations for degeneracy maps in the 2-truncated simplex category. --
abbrev  $\sigma_2$  {n} (i : Fin (n + 1)) (hn := by decide) (hn' := by decide) :
  ([n+1], hn) : SimplexCategory.Truncated 2) → ([n], hn') := SimplexCategory. $\sigma$  i

@[reassoc (attr := simp)]
lemma  $\delta_2\_zero\_comp\_sigma_2\_zero$  {n} (hn := by decide) (hn' := by decide) :
   $\delta_2$  (n := n) 0 hn hn' >>  $\sigma_2$  0 hn' hn = 1 _ := SimplexCategory. $\delta\_comp\_sigma\_self$ 

@[reassoc]
lemma  $\delta_2\_zero\_comp\_sigma_2\_one$  :  $\delta_2$  (0 : Fin 3) >>  $\sigma_2$  1 =  $\sigma_2$  0 >>  $\delta_2$  0 :=
  SimplexCategory. $\delta\_comp\_sigma\_of\_le$  (i := 0) (j := 0) (Fin.zero_le _)

@[reassoc (attr := simp)]
lemma  $\delta_2\_one\_comp\_sigma_2\_zero$  {n} (hn := by decide) (hn' := by decide) :
   $\delta_2$  (n := n) 1 hn hn' >>  $\sigma_2$  0 hn' hn = 1 _ := SimplexCategory. $\delta\_comp\_sigma\_succ$ 

@[reassoc (attr := simp)]
lemma  $\delta_2\_two\_comp\_sigma_2\_one$  :  $\delta_2$  (2 : Fin 3) >>  $\sigma_2$  1 = 1 _ := SimplexCategory. $\delta\_comp\_sigma\_succ'$  (by decide)

@[reassoc]
lemma  $\delta_2\_two\_comp\_sigma_2\_zero$  :  $\delta_2$  (2 : Fin 3) >>  $\sigma_2$  0 =  $\sigma_2$  0 >>  $\delta_2$  1 :=
  SimplexCategory. $\delta\_comp\_sigma\_of\_gt'$  (by decide)
```

## Dependent equality and “evil”



Universal properties characterize objects of a category only up to (unique) isomorphism. Categorical statements that refer to an equality between objects are known as “evil.”

Equality between parallel functors  $F, G: C \Rightarrow D$  is certainly evil, though strangely the “evil” equality between objects is the easiest part to formalize.

## Dependent equality and “evil”



Universal properties characterize objects of a category only up to (unique) isomorphism. Categorical statements that refer to an equality between objects are known as “evil.”

Equality between parallel functors  $F, G: C \Rightarrow D$  is certainly evil, though strangely the “evil” equality between objects is the easiest part to formalize.

On paper,  $F = G$  just when

- $FX = GX$  for all objects  $X$  in  $C$ , and
- $Ff = Gf$  for all arrows  $f: X \rightarrow Y$  in  $C$ .

But **Mathlib**, the second statement does not type check: it is not meaningful to ask whether  $Ff$  equals  $Gf$  because  $Ff: FX \rightarrow FY$  and  $Gf: GX \rightarrow GY$ . Even if we have proofs  $hX: FX = GX$  and  $hY: FY = GY$ , these are different types.

## Dependent equality and “evil”



Using an evil hypothesis  $h\_obj : \forall X, F.obj\ X = G.obj\ X$  involving parallel functors  $F, G : C \Rightarrow D$ , there are various ways to conclude that  $F = G$ :

```
/-- Proving equality between functors. This isn't an extensionality lemma,  
    because usually you don't really want to do this. -/  
theorem ext {F G : C  $\Rightarrow$  D} (h_obj :  $\forall X, F.obj\ X = G.obj\ X$ )  
  (h_map :  $\forall X\ Y\ f,$   
    F.map f = eqToHom (h_obj X)  $\gg$  G.map f  $\gg$  eqToHom (h_obj Y).symm := by aesop_cat) :  
  F = G := by
```

```
/-- Proving equality between functors using heterogeneous equality. -/  
theorem hext {F G : C  $\Rightarrow$  D} (h_obj :  $\forall X, F.obj\ X = G.obj\ X$ )  
  (h_map :  $\forall (X\ Y)\ (f : X \rightarrow Y),$  HEq (F.map f) (G.map f)) : F = G :=
```

```
lemma ext_of_iso {F G : C  $\Rightarrow$  D} (e : F  $\cong$  G) (hobj :  $\forall X, F.obj\ X = G.obj\ X$ )  
  (happ :  $\forall X, e.hom.app\ X = eqToHom\ (hobj\ X)$ ) : F = G :=
```

None of these are particularly fun to use,  
as they involve arguments that are totally invisible on paper.

## Coherence theorems



On paper, a **coherence theorem** of Power tells us that 2-cells in a 2-category compose by pasting, generalization the primitive operations of **vertical composition** and **whiskering**:

$$\begin{array}{ccccccc} & & A & \xrightarrow{G_1} & C & \xlongequal{\quad} & C & \xrightarrow{G_2} & E & \xlongequal{\quad} & E \\ & \nearrow R_1 & \downarrow L_1 & \searrow \alpha & \downarrow L_2 & \nearrow R_2 & \downarrow L_2 & \searrow \beta & \downarrow L_3 & \nearrow R_3 \\ B & \xlongequal{\quad} & B & \xrightarrow{H_1} & D & \xlongequal{\quad} & D & \xrightarrow{H_2} & F & \\ & \searrow \epsilon_1 & & & \downarrow \epsilon_2 & & & & & \end{array}$$

In Mathlib, the 2-cells displayed here belong to dependent types (over their boundary 1-cells and objects).

Depending on how the whiskerings are chosen, 2-cells that are composable on paper are composable in Lean as 1-cells along their common boundary are not definitionally equal:

e.g., is  $R_3 H_2 L_2 \eta_2 G_1 R_1$  composable with  $R_3 H_2 \epsilon_2 L_2 G_1 R_1$ ?



# Coherence theorems



$$\begin{array}{ccccccc} & A & \xrightarrow{G_1} & C & \xlongequal{\quad} & C & \xrightarrow{G_2} & E & \xlongequal{\quad} & E \\ R_1 \nearrow & \downarrow L_1 & & \downarrow L_2 & \nearrow R_2 & \downarrow L_2 & & \downarrow L_3 & \nearrow R_3 \\ B & \xlongequal{\quad} & B & \xrightarrow{H_1} & D & \xlongequal{\quad} & D & \xrightarrow{H_2} & F \end{array}$$

Diagram illustrating a coherence theorem involving 2-cells  $\eta_2$  and  $\eta_3$ , and 1-cells  $G_1, G_2, H_1, H_2$ . The diagram shows a complex composition of 1-cells and 2-cells, with various naturality and coherence conditions indicated by the arrows and labels.

is  $R_3 H_2 L_2 \eta_2 G_1 R_1$   
composable  
with  $R_3 H_2 \epsilon_2 L_2 G_1 R_1$ ?

Lean has a clever composition operation for 2-cells in a bicategory:

```
-- Construct an isomorphism between two objects in a bicategorical category
out of unitors and associators. -/
abbrev bicategoricalIso (f g : a → b) [BicategoricalCoherence f g] : f ≅ g :=
  ⊗ 1

-- Compose two morphisms in a bicategorical category,
inserting unitors and associators between as necessary. -/
def bicategoricalComp {f g h i : a → b} [BicategoricalCoherence g h]
  (η : f → g) (θ : h → i) : f → i :=
  η >> ⊗ 1.hom >> θ
```

but no normal form for whiskered 2-cells or pasting diagram composites.

# Coherence theorems



```
/-- The mates equivalence commutes with vertical composition. -/
theorem mateEquiv_vcomp (α : g₁ » l₂ → l₁ » h₁) (β : g₂ » l₃ → l₂ » h₂) :
  mateEquiv adj₁ adj₂ (leftAdjointSquare.vcomp α β) =
    rightAdjointSquare.vcomp (mateEquiv adj₁ adj₂ α) (mateEquiv adj₂ adj₃ β) := by
  dsimp only [leftAdjointSquare.vcomp, mateEquiv_apply, rightAdjointSquare.vcomp]
  symm
  calc
  _ = 1_ >>> r₁ < g₁ < adj₂.unit > g₂ >>> r₁ < α > r₂ > g₂ >>>
    ((adj₁.counit > (h₁ » r₂ » g₂ » 1 e)) >> 1 b < (h₁ < r₂ < g₂ < adj₃.unit)) >>>
      h₁ < r₂ < β > r₃ >>> h₁ < adj₂.counit > h₂ > r₃ >>> 1_ := by
    bicategory
  _ = 1_ >>> r₁ < g₁ < adj₂.unit > g₂ >>>
    (r₁ < (α > (r₂ » g₂ » 1 e)) > (l₁ » h₁) < r₂ < g₂ < adj₃.unit)) >>>
      ((adj₁.counit > (h₁ » r₂) > (g₂ » l₃) > (1 b » h₁ » r₂) < β) > r₃) >>>
        h₁ < adj₂.counit > h₂ > r₃ >>> 1_ := by
    rw [← whisker_exchange]
    bicategory
  _ = 1_ >>> r₁ < g₁ < (adj₂.unit > (g₂ » 1 e)) > (l₂ » r₂) < g₂ < adj₃.unit) >>>
    (r₁ < (α > (r₂ » g₂ » l₃) > (l₁ » h₁) < r₂ < β) > r₃) >>>
      (adj₁.counit > h₁ > (r₂ » l₂) > (1 b » h₁) < adj₂.counit) > h₂ > r₃ >>> 1_ := by
    rw [← whisker_exchange, ← whisker_exchange]
    bicategory
  _ = 1_ >>> r₁ < g₁ < g₂ < adj₃.unit >>>
    r₁ < g₁ < (adj₂.unit > (g₂ » l₃) > (l₂ » r₂) < β) > r₃ >>>
      r₁ < (α > (r₂ » l₂) > (l₁ » h₁) < adj₂.counit) > h₂ > r₃ >>>
        adj₁.counit > h₁ > h₂ > r₃ >>> 1_ := by
    rw [← whisker_exchange, ← whisker_exchange, ← whisker_exchange]
    bicategory
  _ = 1_ >>> r₁ < g₁ < g₂ < adj₃.unit >>> r₁ < g₁ < β > r₃ >>>
    ((r₁ » g₁) < leftZigzag adj₂.unit adj₂.counit > (h₂ » r₃)) >>>
      r₁ < α > h₂ > r₃ >>> adj₁.counit > h₁ > h₂ > r₃ >>> 1_ := by
    rw [← whisker_exchange, ← whisker_exchange]
    bicategory
  _ = _ := by
    rw [adj₂.left_triangle]
    bicategory
```

A formal proof by [Yuma Mizuno](#) leveraged his bicategory tactic to prove an equality between the previous pasting composite and a reduced form (with the whiskered composite of  $\eta_2$  and  $\epsilon_2$  replaced by an identity).

But his proof required a lot of intermediate calculation — specifying a particular sequence of presentations of the pasted composite as a vertical composite of whiskered 2-cells — that ideally would be automated.



Because the simplex category  $\Delta$  is a **strict Reedy category**, there is a well understood procedure for extending a map  $f: X \rightarrow Y$  of  $n - 1$ -truncated simplicial sets to a map of  $n$ -truncated simplicial sets:

- (i) define the new component:  $f_n: X_n \rightarrow Y_n$ , a function carrying  $n$ -simplices of  $X$  to  $n$ -simplices of  $Y$ , and
- (ii) check that the map  $f_n$  respects degeneracies and faces.

If desired, each of the checks mentioned in (ii) can be encoded as a single commutative square, one involving a colimit forming the **latching object** of degenerate simplices and the other involving a limit forming the **matching object** of simplex face data.



Because the simplex category  $\Delta$  is a **strict Reedy category**, there is a well understood procedure for extending a map  $f: X \rightarrow Y$  of  $n - 1$ -truncated simplicial sets to a map of  $n$ -truncated simplicial sets:

- (i) define the new component:  $f_n: X_n \rightarrow Y_n$ , a function carrying  $n$ -simplices of  $X$  to  $n$ -simplices of  $Y$ , and
- (ii) check that the map  $f_n$  respects degeneracies and faces.

If desired, each of the checks mentioned in (ii) can be encoded as a single commutative square, one involving a colimit forming the **latching object** of degenerate simplices and the other involving a limit forming the **matching object** of simplex face data.

These conditions are both necessary and sufficient.



Mathlib does not know any Reedy category theory. Do we digress to formalize this general abstract nonsense or instead just check that the given family of maps

$$f_0: X_0 \rightarrow Y_0, \quad f_1: X_1 \rightarrow Y_1, \quad \dots, \quad f_{n-1}: X_{n-1} \rightarrow Y_{n-1}, \quad f_n: X_n \rightarrow Y_n$$

defines a natural transformation in the case of interest, checking a lot of unnecessary naturality conditions?

---

<sup>2</sup>While integrating this into Mathlib, Joël Riou provided infrastructure that allowed us to reduce to four cases, though there should have just been two.



Mathlib does not know any Reedy category theory. Do we digress to formalize this general abstract nonsense or instead just check that the given family of maps

$$f_0: X_0 \rightarrow Y_0, \quad f_1: X_1 \rightarrow Y_1, \quad \dots, \quad f_{n-1}: X_{n-1} \rightarrow Y_{n-1}, \quad f_n: X_n \rightarrow Y_n$$

defines a natural transformation in the case of interest, checking a lot of unnecessary naturality conditions?

As mentioned previously, in order to define the unit of the adjunction between the nerve functor and the homotopy category functor, we had to extend a map of 1-truncated simplicial sets to a map of 2-truncated simplicial sets, which requires 31 naturality checks. We essentially did this by hand, splitting into 9 cases, parametrized by pairs of objects.<sup>2</sup>

---

<sup>2</sup>While integrating this into Mathlib, Joël Riou provided infrastructure that allowed us to reduce to four cases, though there should have just been two.

## Conclusion



- There was quite a lot of previous work in Mathlib we were able to build on

## Conclusion



- There was quite a lot of previous work in Mathlib we were able to build on
- This project couldn't have happened without our collaboration; we had some pair programming sessions where Emily would dictate the math to me and I would type it in and we both learned a lot in the process



## Conclusion



- There was quite a lot of previous work in Mathlib we were able to build on
- This project couldn't have happened without our collaboration; we had some pair programming sessions where Emily would dictate the math to me and I would type it in and we both learned a lot in the process
- There are still a lot of friction points before we can get to the point that we could just type in the informal proof directly. Needs more automation?

# Conclusion



- There was quite a lot of previous work in Mathlib we were able to build on
- This project couldn't have happened without our collaboration; we had some pair programming sessions where Emily would dictate the math to me and I would type it in and we both learned a lot in the process
- There are still a lot of friction points before we can get to the point that we could just type in the informal proof directly. Needs more automation?
- The  $\infty$ -cosmos project is underway, so if you like  $\infty$ -categories go check out <https://emilyriehl.github.io/infinity-cosmos/>

## Conclusion



- There was quite a lot of previous work in Mathlib we were able to build on
- This project couldn't have happened without our collaboration; we had some pair programming sessions where Emily would dictate the math to me and I would type it in and we both learned a lot in the process
- There are still a lot of friction points before we can get to the point that we could just type in the informal proof directly. Needs more automation?
- The  $\infty$ -cosmos project is underway, so if you like  $\infty$ -categories go check out <https://emilyriehl.github.io/infinity-cosmos/>

Questions?