Formalizing Category Theory in Lean

AGMSC 2024

Jack McKoen

University of Alberta

July 3, 2024

Outline

Introduction

Lean

Lean is an interactive theorem prover.

Lean

Lean is an interactive theorem prover.

Lean's library of mathematics is called `mathlib`.

Lean

Lean is an interactive theorem prover.

Lean's library of mathematics is called `mathlib`.

`mathlib` has a lot of category theory.

Lean

Lean is based on *dependent type theory*.

Lean

Lean is based on *dependent type theory*.

For the purposes of this talk, any instance of `Type` may be interpreted as Set.

(some of the following code had been modified from `mathlib` for readability.)

Categories

```
class Category (obj : Type u) : Type (max u (v + 1)) where
  id : ∀ (X : obj), (X → X)
  comp : ∀ {X Y Z : obj}, (X → Y) → (Y → Z) → (X → Z)
  id_comp : ∀ {X Y : obj} (f : X → Y), (id X) ≫ f = f
  comp_id : ∀ {X Y : obj} (f : X → Y), f ≫ (id Y) = f
  assoc : ∀ {W X Y Z : obj}
    (f : W → X) (g : X → Y) (h : Y → Z),
    (f ≫ g) ≫ h = f ≫ g ≫ h
```

Functors & Natural Transformations

```
variable (C D : Type u) [Category.{v} C] [Category.{v} D]

structure Functor : Type (max u v)
  obj : C → D
  map : ∀ {X Y : C}, (X ⟶ Y) → (obj X ⟶ obj Y)
  map_id : ∀ (X : C), map (id X) = id (obj X)
  map_comp : ∀ {X Y Z : C} (f : X ⟶ Y) (g : Y ⟶ Z),
    map (f ≫ g) = map f ≫ map g

structure NatTrans (F G : C ⥤ D) : Type (max u v) where
  component : ∀ (X : C), (F.obj X ⟶ G.obj X)
  naturality : ∀ {X Y : C} (f : X ⟶ Y),
    F.map f ≫ (component Y) = (component X) ≫ G.map f
```

The Goal

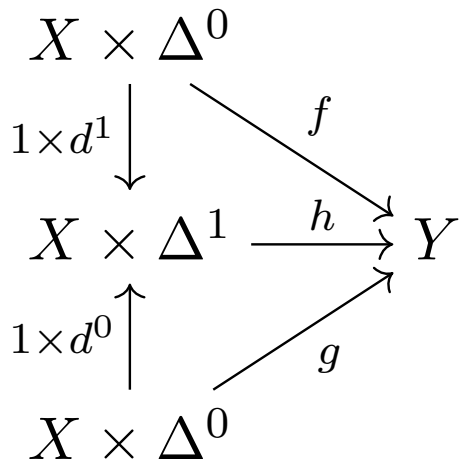
Simplicial Homotopy

I wanted to formalize the concept of a simplicial homotopy.

Simplicial Homotopy

I wanted to formalize the concept of a simplicial homotopy.

Morphisms $f, g : X \to Y$ of simplicial sets are *homotopic* if there exists some $h : X \times \Delta^1 \to Y$ such that the following diagram commutes:



i.e. "$h(x, 0) = f(x)$, $h(x, 1) = g(x)$."

Simplicial Sets

Simplicial Sets

$\mathbf{\Delta}$ is the simplex category, whose objects are finite, totally ordered sets

$$[n] = \{0, 1, \ldots, n\}$$

indexed by $\mathbb{N}$, and whose morphisms are order-preserving functions.

Simplicial Sets

$\mathbf{\Delta}$ is the simplex category, whose objects are finite, totally ordered sets

$$[n] = \{0, 1, \ldots, n\}$$

indexed by $\mathbb{N}$, and whose morphisms are order-preserving functions.

A simplicial object in a category $\mathcal{C}$ is a functor $\mathbf{\Delta}^{\mathrm{op}} \to \mathcal{C}$.

Simplicial Sets

$\mathbf{\Delta}$ is the simplex category, whose objects are finite, totally ordered sets

$$[n] = \{0, 1, \ldots, n\}$$

indexed by $\mathbb{N}$, and whose morphisms are order-preserving functions.

A simplicial object in a category $\mathcal{C}$ is a functor $\mathbf{\Delta}^{\mathrm{op}} \to \mathcal{C}$.

A simplicial set is a simplicial object in **Set**. (in Lean we say it is a simplicial object in `Type`.)

The category of simplicial sets is **SSet**.

Simplicial Sets

According to `mathlib`...

```
def SimplexCategory := ℕ

def SimplicialObject (C : Type u) [Category.{v} C] :=
  SimplexCategory^op ⇒ C

def SSet : Type (u + 1) :=
  SimplicialObject (Type u)
```

The Standard Simplex

The standard n-simplex Δ^n is a simplicial set defined by the Yoneda embedding of $[n] \in \mathbf{\Delta}$:

$$\Delta^n := \mathrm{Hom}_{\mathbf{\Delta}}(-, [n]).$$

The Standard Simplex

The standard n-simplex Δ^n is a simplicial set defined by the Yoneda embedding of $[n] \in \mathbf{\Delta}$:

$$\Delta^n := \mathrm{Hom}_{\mathbf{\Delta}}(-, [n]).$$

We visualize the standard n-simplex as the usual geometric n-simplex: Δ^0 is a point, Δ^1 is a line segment, Δ^2 is a triangle, etc.

The Standard Simplex

The standard n-simplex Δ^n is a simplicial set defined by the Yoneda embedding of $[n] \in \mathbf{\Delta}$:

$$\Delta^n := \mathrm{Hom}_{\mathbf{\Delta}}(-, [n]).$$

We visualize the standard n-simplex as the usual geometric n-simplex: Δ^0 is a point, Δ^1 is a line segment, Δ^2 is a triangle, etc.

```
def standardSimplex : SimplexCategory ⇒ SSet :=
  yoneda
```

The Standard Simplex

For a simplicial set X and $[n] \in \mathbf{\Delta}$, we denote $X[n]$ by X_n and call this set the n-simplices of X.

The Standard Simplex

For a simplicial set X and $[n] \in \mathbf{\Delta}$, we denote $X[n]$ by X_n and call this set the n-simplices of X.

By the Yoneda lemma, X_n is in bijection with morphisms $\Delta^n \to X$:

$$\mathrm{Hom}_{\mathbf{SSet}}(\Delta^n, X) \simeq X_n$$

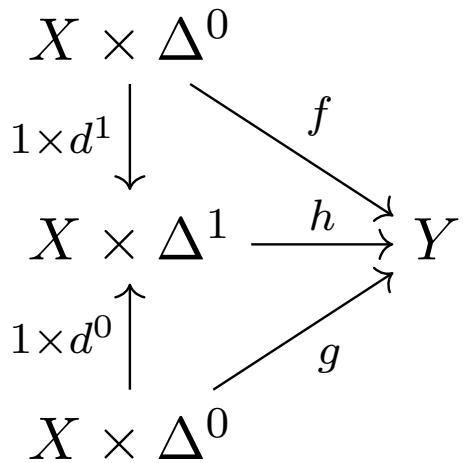
```
def SSet.yonedaEquiv (X : SSet) (n : SimplexCategoryᵒᵖ) :
    (Δ[n] → X) ≃ X.obj n :=
  yonedaCompUliftFunctorEquiv X n
```

Initial Formalizations

Simplicial Homotopy

I wanted to formalize the concept of a simplicial homotopy.

Morphisms $f, g : X \to Y$ of simplicial sets are *homotopic* if there exists some $h : X \times \Delta^1 \to Y$ such that the following diagram commutes:

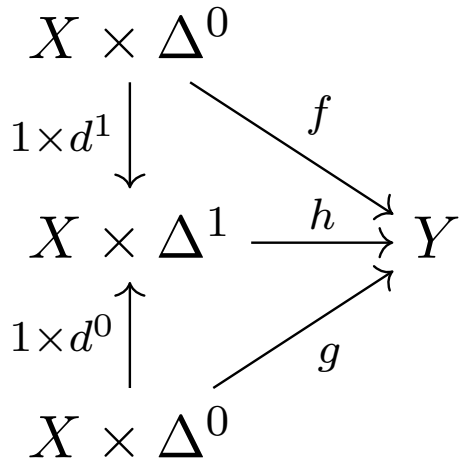


i.e. "$h(x, 0) = f(x)$, $h(x, 1) = g(x)$."

Simplicial Homotopy

At face value this is straightforward, involving only the following:

- morphisms of simplicial sets
- the standard simplex
- products of simplicial sets



Products

The product of simplicial sets X and Y is given by

$$(X \times Y)_n = X_n \times Y_n.$$

Products

The product of simplicial sets X and Y is given by

$$(X \times Y)_n = X_n \times Y_n.$$

Problem: Lean doesn't know this.

Limits

```
variable (C D : Type u) [Category.{v} C D]

/-- `C` has all (small) limits if it has limits of every
    shape that is as big as its hom-sets. -/
class HasLimits : Prop := HasLimitsOfSize.{v, v} C
```

Limits

```
variable (C D : Type u) [Category.{v} C D]

/-- `C` has all (small) limits if it has limits of every
    shape that is as big as its hom-sets. -/
class HasLimits : Prop := HasLimitsOfSize.{v, v} C

instance functorCategoryHasLimits [HasLimits C] :
  HasLimits (D ⇒ C) := inferInstance
```

Limits

```
variable (C D : Type u) [Category.{v} C D]

/-- `C` has all (small) limits if it has limits of every
    shape that is as big as its hom-sets. -/
class HasLimits : Prop := HasLimitsOfSize.{v, v} C

instance functorCategoryHasLimits [HasLimits C] :
  HasLimits (D ⇒ C) := inferInstance

instance [HasLimits C] : HasLimits (SimplicialObject C) :=
  inferInstance
```

Products

Lean knows that **SSet** has all limits, so it knows that **SSet** has products.

But it has no idea what this product looks like, because it's inherited from some super general categorical nonsense.

Products

Lean knows that **SSet** has all limits, so it knows that **SSet** has products.

But it has no idea what this product looks like, because it's inherited from some super general categorical nonsense.

This isn't a problem until you want to do explicit things with products (like we do).

Products

```
def prod (X Y : SSet) : SSet where
  obj n := X.obj n × Y.obj n
  map f n := (X.map f n.1, Y.map f n.2)

noncomputable def binaryProductIso : X × Y ≅ prod X Y :=
  limit.isoLimitCone (binaryProductLimitCone X Y)
```

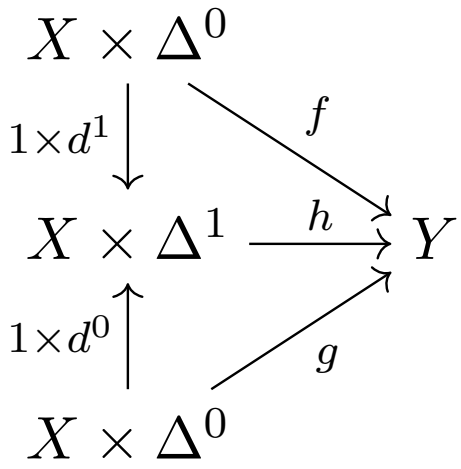
Coproducts

```
def coprod (X Y : SSet) : SSet where
  obj n := X.obj n ⨁ X.obj n
  map f n := by
    cases x with
    | inl n => exact .inl (X.map f n)
    | inr n => exact .inr (Y.map f n)

noncomputable def binaryCoproductIso : X ⨿ Y ≅ coprod X Y :=
  colimit.isoColimitCocone (binaryCoproductColimitCocone X Y)
```

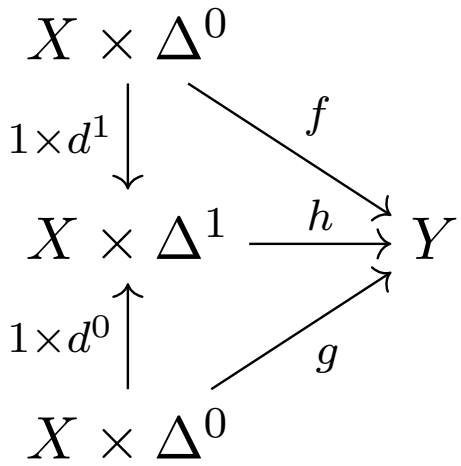
Back to Homotopy

We have workable products now.



Back to Homotopy

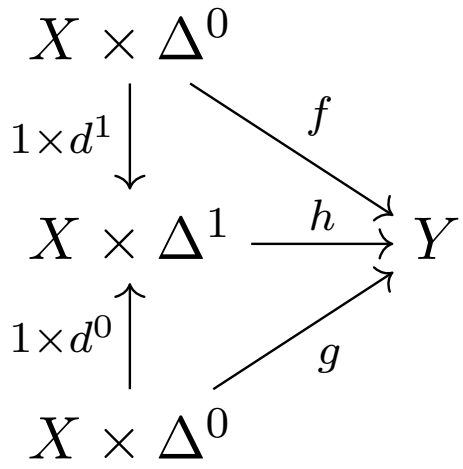
We have workable products now.



Recall that $f, g : X \to Y$, so in the diagram above we implicitly use that $X \times \Delta^0 \cong X$.

Back to Homotopy

We have workable products now.

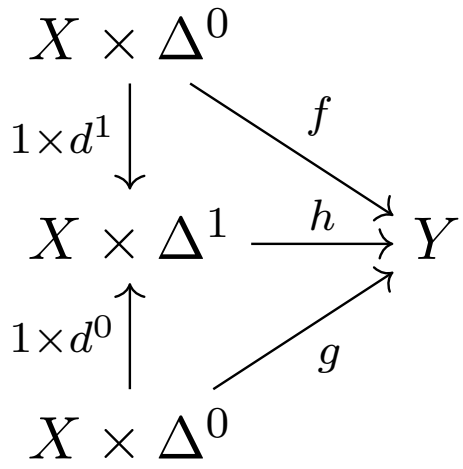


Recall that $f, g : X \to Y$, so in the diagram above we implicitly use that $X \times \Delta^0 \cong X$.

Why does this isomorphism exist?

Back to Homotopy

We have workable products now.



Recall that $f, g : X \to Y$, so in the diagram above we implicitly use that $X \times \Delta^0 \cong X$.

Why does this isomorphism exist?

We get this isomorphism for free if we talk about the monoidal category structure on **SSet**.

Toward Generality

Monoidal Categories

SSet is a *monoidal category*.

Monoidal Categories

SSet is a *monoidal category*.

This means it has structure resembling a *monoid*: we can define a "binary operation" (called the tensor product) on simplicial sets which is associative and unital.

Monoidal Categories

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

class MonoidalCategory where
  tensorProd : C → C → C -- denote by ⊗
  tensorUnit : C -- denote by 𝟙
  associator : ∀ (X Y Z : C), (X ⊗ Y) ⊗ Z ≅ X ⊗ (Y ⊗ Z)
  leftUnitor : ∀ (X : C), (tensorUnit ⊗ X) ≅ X
  rightUnitor : ∀ (X : C), (X ⊗ tensorUnit) ≅ X
  pentagon :
    /- The identity relating the isomorphism between
       `X ⊗ (Y ⊗ (Z ⊗ W))` and `((X ⊗ Y) ⊗ Z) ⊗ W` -/
  triangle :
    /- The identity relating the isomorphisms between
       `X ⊗ (𝟙 ⊗ Y)`, `(X ⊗ 𝟙) ⊗ Y`,
       and `X ⊗ Y` -/
```

Monoidal Categories

SSet has finite products, so in particular it has the empty product, which is the terminal object (in this case, Δ^0).

Monoidal Categories

SSet has finite products, so in particular it has the empty product, which is the terminal object (in this case, Δ^0).

Categories with finite products naturally form a *monoidal category*, with

$$X \otimes Y := X \times Y$$

and

$$\mathbb{1} := \text{the terminal object.}$$

Monoidal Categories

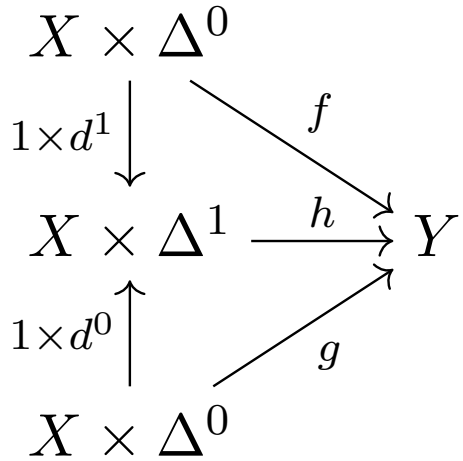
```
instance : HasFiniteProducts SSet := inferInstance

def monoidalOfHasFiniteProducts [HasFiniteProducts C] :
    MonoidalCategory C :=
  tensorProd := fun X Y ↦ X × Y
  tensorUnit := terminal C
  associator := prod.associator
  leftUnitor := fun P ↦ prod.leftUnitor P
  rightUnitor := fun P ↦ prod.rightUnitor P
```

In **SSet**, `rightUnitor` provides $X \times \Delta^0 \cong X$.

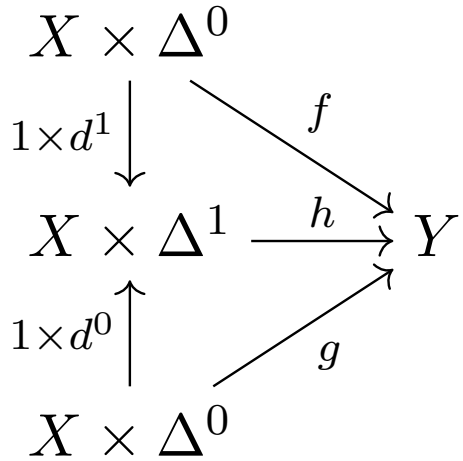
Monoidal Categories

So we defined a monoidal structure on **SSet** and now we have the desired isomorphism $X \times \Delta^0 \cong X$.



Monoidal Categories

So we defined a monoidal structure on **SSet** and now we have the desired isomorphism $X \times \Delta^0 \cong X$.



But why stop here?

More Generality

Monoidal Closed Categories

SSet is actually a *monoidal closed* category.

Monoidal Closed Categories

SSet is actually a *monoidal closed* category.

This means that for every simplicial set Y, the functor

$$(-) \times Y : \mathbf{SSet} \to \mathbf{SSet}$$

admits a right adjoint

$$[Y, -] : \mathbf{SSet} \to \mathbf{SSet}.$$

We call $[Y, Z]$ the *internal hom* of Y and Z.

Monoidal Closed Categories

```
variable (C : Type u) [Category.{v} C]
variable [MonoidalCategory.{v} C]

class MonoidalClosed where
  /-- a choice of a right adjoint for `(-) ⊗ Y` -/
  rightAdj : (∀ Y : C), C ⇒ C
  /-- `(-) ⊗ Y` is a left adjoint -/
  adj : (∀ Y : C), tensorRight Y ⊣ rightAdj

def ihom (Y Z : C) : C ⇒ C :=
  (rightAdj Y).obj Z
```

Internal Homs

In **SSet**, the internal hom is given by

$$[Y, Z]_n := \mathrm{Hom}(\Delta^n \times Y, Z).$$

The fact that this is an adjoint follows directly from the Yoneda lemma:

$$\mathrm{Hom}(\Delta^n, [Y, Z]) \simeq [Y, Z]_n = \mathrm{Hom}(\Delta^n \times Y, Z).$$

```
def ihom (X Y : SSet) : SSet where
  obj n := Δ[n] × X → Y
  map f g := standardSimplex.map f  F ≫ g
```

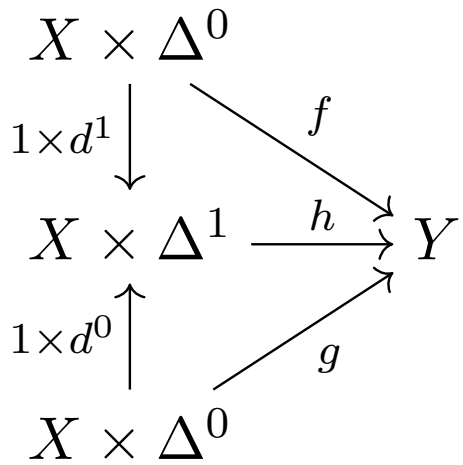
Where Am I Going With This?

Given simplicial sets X and Y, we've defined a new simplicial set $[X, Y]$, the internal hom.

Where Am I Going With This?

Given simplicial sets X and Y, we've defined a new simplicial set $[X, Y]$, the internal hom.

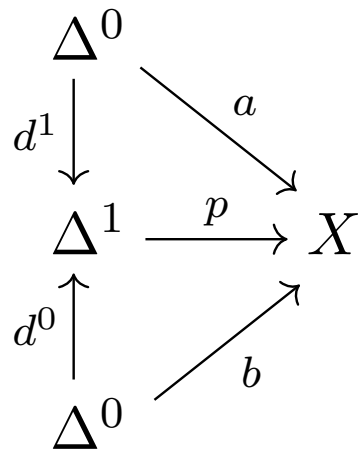
Why did we bother doing that?



Paths

A *path* in a simplicial set X is a 1-simplex $p \in X_1$. For 0-simplices $a, b \in X_0$, say p is a path from a to b if the following diagram commutes:

(recall that $X_n \simeq \mathrm{Hom}_{\mathbf{SSet}}(\Delta^n, X)$ by the Yoneda lemma.)



i.e. "$p(0) = a$, $p(1) = b$."

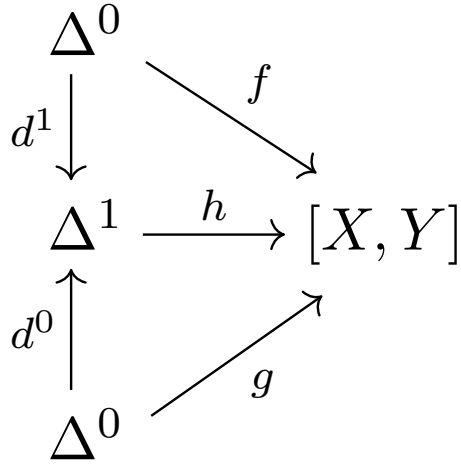
Paths

What is a path in $[X, Y]$? Let

$$h \in [X, Y]_1 = \mathrm{Hom}(\Delta^1 \times X, Y),$$

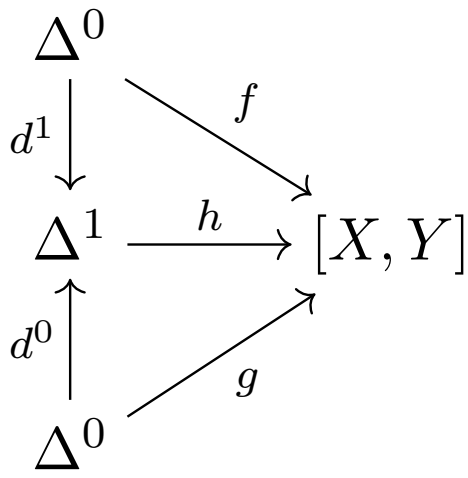
and

$$f, g \in [X, Y]_0 = \mathrm{Hom}(\Delta^0 \times X, Y) \simeq \mathrm{Hom}(X, Y).$$

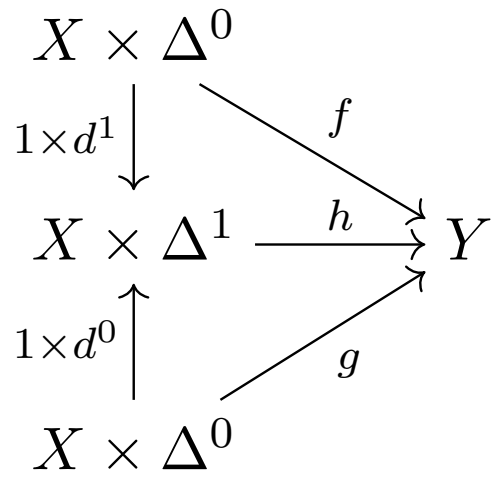


Back to Homotopy

It is then straightforward to show that a simplicial homotopy is just a path in $[X, Y]$:



path



homotopy

Conclusion

Summary

- we defined explicit (co)products of simplicial sets

Summary

- we defined explicit (co)products of simplicial sets
- using this product, we defined a monoidal category structure on **SSet**

Summary

- we defined explicit (co)products of simplicial sets
- using this product, we defined a monoidal category structure on **SSet**
- we showed that this monoidal structure is closed

Summary

- we defined explicit (co)products of simplicial sets
- using this product, we defined a monoidal category structure on **SSet**
- we showed that this monoidal structure is closed
- we used the resulting internal hom to define simplicial homotopies via paths

Summary

- we defined explicit (co)products of simplicial sets
- using this product, we defined a monoidal category structure on **SSet**
- we showed that this monoidal structure is closed
- we used the resulting internal hom to define simplicial homotopies via paths
- we built a ton of API along the way

Are We Done?

Are We Done?

No, we are not done.

Are We Done?

No, we are not done.

Most of what we proved is true more generally for functors $\mathcal{C} \to \mathbf{Set}$.

Are We Done?

No, we are not done.

Most of what we proved is true more generally for functors $\mathcal{C} \to \mathbf{Set}$.

We can also generalize homotopies further with *enriched categories*.

Are We Done?

No, we are not done.

Most of what we proved is true more generally for functors $\mathcal{C} \to \mathbf{Set}$.

We can also generalize homotopies further with *enriched categories*.

A *simplicial category* is a category $\mathcal{C}$ that is enriched over **SSet** in such a way that morphisms in $\mathcal{C}$ identify to the 0-simplices of the enriched hom.

Are We Done?

No, we are not done.

Most of what we proved is true more generally for functors $\mathcal{C} \to \mathbf{Set}$.

We can also generalize homotopies further with *enriched categories*.

A *simplicial category* is a category $\mathcal{C}$ that is enriched over **SSet** in such a way that morphisms in $\mathcal{C}$ identify to the 0-simplices of the enriched hom.

We can then define the notion of homotopy between morphisms in any simplicial category.

Are We Done?

No, we are not done.

Most of what we proved is true more generally for functors $\mathcal{C} \to \mathbf{Set}$.

We can also generalize homotopies further with *enriched categories*.

A *simplicial category* is a category $\mathcal{C}$ that is enriched over **SSet** in such a way that morphisms in $\mathcal{C}$ identify to the 0-simplices of the enriched hom.

We can then define the notion of homotopy between morphisms in any simplicial category.

SSet is a simplicial category, and in this case we recover the definition of homotopy we just came up with.

References

Emily Riehl

A Leisurely Introduction To Simplicial Sets, 2011