Formalizing Category Theory in Agda

Jason Hu Jacques Carette

McGill University McMaster University

Introduction

Why do we formalize category theory in proof assistants?

Introduction

Why do we formalize category theory in proof assistants?

- study category theory;

Introduction

Why do we formalize category theory in proof assistants?

- study category theory;
- study the proof assistant;

Introduction

Why do we formalize category theory in proof assistants?

- study category theory;
- study the proof assistant;
- study other fields using the formalized category theory.

Previous Work

- Various systems: Agda, Coq, Isabelle, Lean, Idris.

Previous Work

- Various systems: Agda, Coq, Isabelle, Lean, Idris.
- In Agda, there was a previous library [Peebles et al., 2018].

Previous Work

- Various systems: Agda, Coq, Isabelle, Lean, Idris.
- In Agda, there was a previous library [Peebles et al., 2018].
- Most development effort is spent on multiple Coq libraries.

Previous Work

- Various systems: Agda, Coq, Isabelle, Lean, Idris.
- In Agda, there was a previous library [Peebles et al., 2018].
- Most development effort is spent on multiple Coq libraries.

libraries	proof assistants	foundation	LoC
[Peebles et al., 2018]	Agda 2.5.2	MLTT + K + irrelevance	11770
[Timany and Jacobs, 2016]	Coq 8.11.1	CIC	14711
[Wiegley, 2019]	Coq 8.10.2	CIC	23003
[Huet and Saïbi, 2000]	Coq 8.12.0	CIC	7879
[Voevodsky et al., , Ahrens et al., 2015]	Coq 8.12.0	HoTT	96366
[Gross et al., 2014]	Hoq 8.12	HoTT with HIT	10604
[mathlib Community, 2020]	Lean	CIC	14975
[Stark, 2016, Stark, 2017, Stark, 2020]	Isabelle	HOL	82782

Our Motivation

- The previous library [Peebles et al., 2018] stopped working:
  - only compiled up to Agda 2.5.2
  - furthermore used Streicher's axiom K, irrelevance and postulates

Our Motivation

- The previous library [Peebles et al., 2018] stopped working:
  - only compiled up to Agda 2.5.2
  - furthermore used Streicher's axiom K, irrelevance and postulates
- The current library rewrites [Peebles et al., 2018]:

Our Motivation

- The previous library [Peebles et al., 2018] stopped working:
  - only compiled up to Agda 2.5.2
  - furthermore used Streicher's axiom K, irrelevance and postulates
- The current library rewrites [Peebles et al., 2018]:
  - Adapt to Agda 2.6+.

Our Motivation

- The previous library [Peebles et al., 2018] stopped working:
  - only compiled up to Agda 2.5.2
  - furthermore used Streicher's axiom K, irrelevance and postulates
- The current library rewrites [Peebles et al., 2018]:
  - Adapt to Agda 2.6+.
  - Limit ourselves to MLTT.

Our Motivation

- The previous library [Peebles et al., 2018] stopped working:
  - only compiled up to Agda 2.5.2
  - furthermore used Streicher's axiom K, irrelevance and postulates
- The current library rewrites [Peebles et al., 2018]:
  - Adapt to Agda 2.6+.
  - Limit ourselves to MLTT.
  - Redesign the module organization

Our Motivation

- The previous library [Peebles et al., 2018] stopped working:
  - only compiled up to Agda 2.5.2
  - furthermore used Streicher's axiom K, irrelevance and postulates
- The current library rewrites [Peebles et al., 2018]:
  - Adapt to Agda 2.6+.
  - Limit ourselves to MLTT.
  - Redesign the module organization
  - Connect with the stdlib more tightly

Our Motivation

- The previous library [Peebles et al., 2018] stopped working:
  - only compiled up to Agda 2.5.2
  - furthermore used Streicher's axiom K, irrelevance and postulates
- The current library rewrites [Peebles et al., 2018]:
  - Adapt to Agda 2.6+.
  - Limit ourselves to MLTT.
  - Redesign the module organization
  - Connect with the stdlib more tightly
  - Prove more properties

What Does It Have?

- ~27000 lines of code (~24000 in artifact) and growing!
    - $\geq$ 10 contributors other than us
    - $\geq$ 2x the code in the previous library

What Does It Have?

- ~27000 lines of code (~24000 in artifact) and growing!
  - $\geq$ 10 contributors other than us
  - $\geq$ 2x the code in the previous library
- Lots of (setoid-based) elementary category theory

What Does It Have?

- ~27000 lines of code (~24000 in artifact) and growing!
  - $\geq$ 10 contributors other than us
  - $\geq$ 2x the code in the previous library
- Lots of (setoid-based) elementary category theory
  - Most of basic definitions

What Does It Have?

- ~27000 lines of code (~24000 in artifact) and growing!
    - $\geq$ 10 contributors other than us
    - $\geq$ 2x the code in the previous library
- Lots of (setoid-based) elementary category theory
    - Most of basic definitions
    - Many lemmas and theorems

What Does It Have?

- ~27000 lines of code (~24000 in artifact) and growing!
  - $\geq$ 10 contributors other than us
  - $\geq$ 2x the code in the previous library
- Lots of (setoid-based) elementary category theory
  - Most of basic definitions
  - Many lemmas and theorems
- A decent amount of enriched category theory and higher category theory

It Implements

Just to name a few (non-exhaustively):

- Concepts:
  - category, functor, natural transformation, adjoint functors;
  - various monoidal categories, cartesian closed category, comma category,
  - initial / terminal, (co)product, (co)end, etc.

It Implements

Just to name a few (non-exhaustively):

- Concepts:
  - category, functor, natural transformation, adjoint functors;
  - various monoidal categories, cartesian closed category, comma category,
  - initial / terminal, (co)product, (co)end, etc.
- Constructions:
  - the category of categories, of set(oid)s, of presheaves, of monoidal categories, etc.

It Implements

Just to name a few (non-exhaustively):

- Concepts:
  - category, functor, natural transformation, adjoint functors;
  - various monoidal categories, cartesian closed category, comma category,
  - initial / terminal, (co)product, (co)end, etc.
- Constructions:
  - the category of categories, of set(oid)s, of presheaves, of monoidal categories, etc.
- Properties:
  - the Yoneda lemma,
  - Freyd's adjoint functor theorem,
  - Lambek's lemma,
  - Right adjoints preserve limits,
  - (local) cartesian closure of `Setoids`,
  - etc.

Basic Design Principles

Try to make the library as general as possible (within type theory):

Basic Design Principles

Try to make the library as general as possible (within type theory):

- Hom-setoids
- universe polymorphism
- *definitional* duality
- records for encapsulation
- predicate versus structure

Hom-setoids and Universe Polymorphism

Basic Design Principles

- Hom-sets as setoids
  - Follow the stdlib and compatible with `--with-K` and `--cubical`.

Hom-setoids and Universe Polymorphism

Basic Design Principles

- Hom-sets as setoids
  - Follow the stdlib and compatible with `--with-K` and `--cubical`.
- Universe polymorphism allows concept reuse across levels.
  - Agda is non-cumulative; cumulativity still buggy now.

Hom-setoids and Universe Polymorphism

Basic Design Principles

- Hom-sets as setoids
  - Follow the stdlib and compatible with `--with-K` and `--cubical`.
- Universe polymorphism allows concept reuse across levels.
  - Agda is non-cumulative; cumulativity still buggy now.

```
record Category (o ℓ e : Level) : Set (suc (o ⊔ ℓ ⊔ e)) where
  field
    Obj : Set o
    _⇒_ : (A B : Obj) → Set ℓ
    _∘_ : ∀ {A B C} → B ⇒ C → A ⇒ B → A ⇒ C

    _≈_ : ∀ {A B} → (f g : A ⇒ B) → Set e
    equiv : ∀ {A B} → IsEquivalence (_≈_ {A} {B})
    -- ignore other laws
```

Duality: Additional Laws

Basic Design Principles

- Duality: “prove one get two”

Duality: Additional Laws

Basic Design Principles

- Duality: "prove one get two"
- Best if duality holds *definitionally*:
  - $(\mathcal{C}^{op})^{op}$ v.s. $\mathcal{C}$
  - $A \times B$ in $\mathcal{C}^{op}$ v.s. $A + B$ in $\mathcal{C}$

Duality: Additional Laws

Basic Design Principles

- Duality: “prove one get two”
- Best if duality holds *definitionally*:
  - $(\mathcal{C}^{op})^{op}$ v.s. $\mathcal{C}$
  - $A \times B$ in $\mathcal{C}^{op}$ v.s. $A + B$ in $\mathcal{C}$
- additional laws might be necessary
  - Category:

```
assoc : (h ∘ g) ∘ f ≈ h ∘ (g ∘ f)
sym-assoc : h ∘ (g ∘ f) ≈ (h ∘ g) ∘ f
```

Duality: Additional Laws

Basic Design Principles

- Duality: "prove one get two"
- Best if duality holds *definitionally*:
  - $(\mathcal{C}^{op})^{op}$ v.s. $\mathcal{C}$
  - $A \times B$ in $\mathcal{C}^{op}$ v.s. $A + B$ in $\mathcal{C}$
- additional laws might be necessary
  - Category:

    ```
    assoc : (h ∘ g) ∘ f ≈ h ∘ (g ∘ f)
    sym-assoc : h ∘ (g ∘ f) ≈ (h ∘ g) ∘ f
    ```
  - Many concepts, e.g. `Monad` and `NaturalTransformation`, require similar additional laws.

Duality: Redundant Definitions

Basic Design Principles

- Dual concepts are defined individually – makes their end-use considerably clearer.
  - products and coproducts
  - monads and comonads

Duality: Redundant Definitions

Basic Design Principles

- Dual concepts are defined individually – makes their end-use considerably clearer.
  - products and coproducts
  - monads and comonads
- Conversions between duals are defined in `*.Duality`.

Duality: Redundant Definitions

Basic Design Principles

- Dual concepts are defined individually – makes their end-use considerably clearer.
  - products and coproducts
  - monads and comonads
- Conversions between duals are defined in `*.Duality`.
  - Help us to ensure we got the definition right.

```
Comonad⇔coMonad : ∀ (M : Comonad C) →
  coMonad⇒Comonad (Comonad⇒coMonad M) ≡ M
Comonad⇔coMonad _ = ≡.refl
```

  The proof body *must be* `≡.refl`.

Records for Encapsulation

Basic Design Principles

- We use records to encode concepts; plays well with the record modules feature of Agda.

Records for Encapsulation

Basic Design Principles

- We use records to encode concepts; plays well with the record modules feature of Agda.
-
```
record Monad {o ℓ e} (C : Category o ℓ e) : Set (o ⊔ ℓ ⊔ e) where
  field
    F : Endofunctor C
    η : NaturalTransformation idF F
```

Records for Encapsulation

Basic Design Principles

- We use records to encode concepts; plays well with the record modules feature of Agda.
-
```
record Monad {o ℓ e} (C : Category o ℓ e) : Set (o ⊔ ℓ ⊔ e) where
  field
    F : Endofunctor C
    η : NaturalTransformation idF F
```

Given a `Monad M`, we can have

```
Functor.F₀ (Monad.F M)
Functor.F₁ (Monad.F M)
NaturalTransformation.commute (Monad.η M) f
```

Records for Encapsulation

Basic Design Principles

- We use records to encode concepts; plays well with the record modules feature of Agda.
- ```
  record Monad {o ℓ e} (C : Category o ℓ e) : Set (o ⊔ ℓ ⊔ e) where
    field
      F : Endofunctor C
      η : NaturalTransformation idF F
    module F = Functor F
    module η = NaturalTransformation η
  ```

  Given a Monad M, we can have

  ```
  Functor.F₀ (Monad.F M)
  Functor.F₁ (Monad.F M)
  NaturalTransformation.commute (Monad.η M) f
  ```
  ```

Records for Encapsulation

Basic Design Principles

- We use records to encode concepts; plays well with the record modules feature of Agda.
-
```
record Monad {o ℓ e} (C : Category o ℓ e) : Set (o ⊔ ℓ ⊔ e) where
  field
    F : Endofunctor C
    η : NaturalTransformation idF F
  module F = Functor F
  module η = NaturalTransformation η
```

  Given a `Monad M`, after declaring `module M = Monad M`, we can have

```
M.F.F₀
M.F.F₁
M.η.commute f
```

Arranging Concepts

Basic Design Principles

- 2 styles for concept representation: predicate versus structure

Arranging Concepts

Basic Design Principles

- 2 styles for concept representation: predicate versus structure
  - Predicate: “is-a” relation;
  - Structure: “has-a” relation.

Arranging Concepts

Basic Design Principles

- 2 styles for concept representation: predicate versus structure
  - Predicate: "is-a" relation;
  - Structure: "has-a" relation.
- This choice is fundamental in systems with automated search machinery.

Arranging Concepts

Basic Design Principles

- 2 styles for concept representation: predicate versus structure
  - Predicate: “is-a” relation;
  - Structure: “has-a” relation.
- This choice is fundamental in systems with automated search machinery.
- In Agda, more about usability and namespace management.

Arranging Concepts

Basic Design Principles

- 2 styles for concept representation: predicate versus structure
  - Predicate: "is-a" relation;
  - Structure: "has-a" relation.
- This choice is fundamental in systems with automated search machinery.
- In Agda, more about usability and namespace management.
- In fact, we see benefits in combining both choices!

```
record Monoidal {o ℓ e}
    (C : Category o ℓ e)
  : Set (o ⊔ ℓ ⊔ e) where
```

```
record MonoidalCategory o ℓ e
    : Set (suc (o ⊔ ℓ ⊔ e)) where
  field
    U : Category o ℓ e
    monoidal : Monoidal U
```

Predicate versus Structure

Basic Design Principles

- Advantage:
  - predicate: properties of one instance of the concept.
  - structure: properties involving multiple instances.

Predicate versus Structure

Basic Design Principles

- Advantage:
  - predicate: properties of one instance of the concept.
  - structure: properties involving multiple instances.
- Use `Monoidal` for properties of one monoidal category.

Predicate versus Structure

Basic Design Principles

- Advantage:
  - predicate: properties of one instance of the concept.
  - structure: properties involving multiple instances.
- Use `Monoidal` for properties of one monoidal category.
- But `MonoidalCategory` is more convenient for:

```
record MonoidalFunctor
  (C : MonoidalCategory o ℓ e)
  (D : MonoidalCategory o′ ℓ′ e′)
  : Set _ where
```

Predicate versus Structure

Basic Design Principles

- Advantage:
  - predicate: properties of one instance of the concept.
  - structure: properties involving multiple instances.
- Use `Monoidal` for properties of one monoidal category.
- But `MonoidalCategory` is more convenient for:

```
record MonoidalFunctor
  (C : MonoidalCategory o ℓ e)
  (D : MonoidalCategory o′ ℓ′ e′)
  : Set _ where
```

```
record MonoidalFunctor′
  {C : Category o ℓ e} {D : Category o′ ℓ′ e′}
  (MC : Monoidal C)    (MD : Monoidal D)
  : Set _ where
```

More Type Theoretic Category Theory

- Category theory on paper is primarily set theoretic:
  - small, locally small, finite, etc

More Type Theoretic Category Theory

- Category theory on paper is primarily set theoretic:
  - small, locally small, finite, etc
- Not quite type theoretic $\Rightarrow$ formalizing as is assumes set theoretic interpretation

More Type Theoretic Category Theory

- Category theory on paper is primarily set theoretic:
  - small, locally small, finite, etc
- Not quite type theoretic $\Rightarrow$ formalizing as is assumes set theoretic interpretation

More type theoretic constructs:

More Type Theoretic Category Theory

- Category theory on paper is primarily set theoretic:
  - small, locally small, finite, etc
- Not quite type theoretic $\Rightarrow$ formalizing as is assumes set theoretic interpretation

More type theoretic constructs:

- Picking type theoretic definitions

More Type Theoretic Category Theory

- Category theory on paper is primarily set theoretic:
  - small, locally small, finite, etc
- Not quite type theoretic $\Rightarrow$ formalizing as is assumes set theoretic interpretation

More type theoretic constructs:

- Picking type theoretic definitions
- natural isos *between Hom-sets* as adjunctions + mates

More Type Theoretic Category Theory

- Category theory on paper is primarily set theoretic:
  - small, locally small, finite, etc
- Not quite type theoretic $\Rightarrow$ formalizing as is assumes set theoretic interpretation

More type theoretic constructs:

- Picking type theoretic definitions
- natural isos *between Hom-sets* as adjunctions + mates
- set theoretic quantification as adjoint equivalence
  - finite categories

Adjoint Functors

- Consider adjoint functors:

```
record Adjoint {C : Category o ℓ e} {D : Category o′ ℓ′ e′}
        (L : Functor C D) (R : Functor D C) : Set _ where
```

Adjoint Functors

- Consider adjoint functors:

```
record Adjoint {C : Category o ℓ e} {D : Category o′ ℓ′ e′}
       (L : Functor C D) (R : Functor D C) : Set _ where
```

- $L \dashv R$ iff $Hom_D(L-,-) \simeq Hom_C(-,R-)$.

Adjoint Functors

- Consider adjoint functors:

```
record Adjoint {C : Category o ℓ e} {D : Category o′ ℓ′ e′}
       (L : Functor C D) (R : Functor D C) : Set _ where
```

- $L \dashv R$ iff $Hom_D(L-, -) \simeq Hom_C(-, R-)$.
- Universe levels of C and D are unrelated $\Rightarrow$ Hom functors cannot be directly related.

Adjoint Functors

- Consider adjoint functors:

```
record Adjoint {C : Category o ℓ e} {D : Category o′ ℓ′ e′}
       (L : Functor C D) (R : Functor D C) : Set _ where
```

- $L \dashv R$ iff $Hom_D(L-,-) \simeq Hom_C(-,R-)$.
- Universe levels of C and D are unrelated $\Rightarrow$ Hom functors cannot be directly related.
  - (Ugly) use lifting functors:

$$Lift \circ Hom_D(L-,-) \simeq Lift \circ Hom_C(-,R-)$$

Adjoint Functors

- Consider adjoint functors:

```
record Adjoint {C : Category o ℓ e} {D : Category o′ ℓ′ e′}
       (L : Functor C D) (R : Functor D C) : Set _ where
```

- $L \dashv R$ iff $Hom_D(L-,-) \simeq Hom_C(-,R-)$.
- Universe levels of C and D are unrelated $\Rightarrow$ Hom functors cannot be directly related.
  - (Ugly) use lifting functors:

$$Lift \circ Hom_D(L-,-) \simeq Lift \circ Hom_C(-,R-)$$

- This form of lifting is required in many definitions / statements in general.

Unit-Counit Definition of Adjoint Functors

- We instead use the unit-counit definition of Adjoint functors:

Definition

Functors $L : \mathcal{C} \Rightarrow \mathcal{D}$ and $R : \mathcal{D} \Rightarrow \mathcal{C}$ are adjoint, $L \dashv R$, if there exist two natural transformations, unit $\eta : 1_{\mathcal{C}} \Rightarrow RL$ and counit $\epsilon : LR \Rightarrow 1_{\mathcal{D}}$, so that the triangle identities below hold:

1. $\epsilon L \circ L\eta = 1_L$
2. $R\epsilon \circ \eta R = 1_R$

Unit-Counit Definition of Adjoint Functors

- We instead use the unit-counit definition of Adjoint functors:

Definition

Functors $L : \mathcal{C} \Rightarrow \mathcal{D}$ and $R : \mathcal{D} \Rightarrow \mathcal{C}$ are adjoint, $L \dashv R$, if there exist two natural transformations, unit $\eta : 1_{\mathcal{C}} \Rightarrow RL$ and counit $\epsilon : LR \Rightarrow 1_{\mathcal{D}}$, so that the triangle identities below hold:

1. $\epsilon L \circ L\eta = 1_L$
2. $R\epsilon \circ \eta R = 1_R$

- Advantage: does not (explicitly) involve any Hom-sets, or universe levels.

Unit-Counit Definition of Adjoint Functors

- We instead use the unit-counit definition of Adjoint functors:

Definition

Functors $L : \mathcal{C} \Rightarrow \mathcal{D}$ and $R : \mathcal{D} \Rightarrow \mathcal{C}$ are adjoint, $L \dashv R$, if there exist two natural transformations, unit $\eta : 1_{\mathcal{C}} \Rightarrow RL$ and counit $\epsilon : LR \Rightarrow 1_{\mathcal{D}}$, so that the triangle identities below hold:

1. $\epsilon L \circ L\eta = 1_L$
2. $R\epsilon \circ \eta R = 1_R$

- Advantage: does not (explicitly) involve any Hom-sets, or universe levels.
- Lesson: unlearn set-theoretic constructs when formalizing categories in type theory!

Mate: Relating Two Adjunctions

- We sometimes need to relate two adjunctions $F \dashv G$ and $F' \dashv G'$:

$$\begin{array}{ccc} Hom(F'X, Y) & \overset{\simeq}{\dashrightarrow} & Hom(X, G'Y) \\ {\scriptstyle Hom(\alpha_X, Y)} \downarrow & & \downarrow {\scriptstyle Hom(X, \beta_Y)} \\ Hom(FX, Y) & \overset{\simeq}{\dashrightarrow} & Hom(X, GY) \end{array}$$

where $\alpha : F \Rightarrow F'$ and $\beta : G' \Rightarrow G$.

Mate: Relating Two Adjunctions

- We sometimes need to relate two adjunctions $F \dashv G$ and $F' \dashv G'$:

$$\begin{array}{ccc} Hom(F'X, Y) & \overset{\simeq}{\cdots\cdots\rightarrow} & Hom(X, G'Y) \\ {\scriptstyle Hom(\alpha_X, Y)}\downarrow & & \downarrow{\scriptstyle Hom(X, \beta_Y)} \\ Hom(FX, Y) & \overset{\simeq}{\cdots\cdots\rightarrow} & Hom(X, GY) \end{array}$$

  where $\alpha : F \Rightarrow F'$ and $\beta : G' \Rightarrow G$.
- It also has a definition purely in terms of morphism equality:

Definition

Two natural transformation $\alpha : F \Rightarrow F'$ and $\beta : G' \Rightarrow G$ form a mate for two pairs of adjunctions $(\eta, \epsilon) : F \dashv G$ and $(\eta', \epsilon') : F' \dashv G'$, if the following two diagrams commute:

$$\begin{array}{ccc} 1_{\mathcal{C}} & \xrightarrow{\eta} & GF \\ {\scriptstyle \eta'}\downarrow & & \downarrow{\scriptstyle G\alpha} \\ G'F' & \xrightarrow{\beta F'} & GF' \end{array} \qquad \begin{array}{ccc} FG' & \xrightarrow{\alpha G'} & F'G' \\ {\scriptstyle F\beta}\downarrow & & \downarrow{\scriptstyle \epsilon'} \\ FG & \xrightarrow{\epsilon} & 1_{\mathcal{D}} \end{array}$$

Example: Closed Monoidal Categories

- We can replace certain natural isomorphisms between Hom-sets with adjoint functors and mates.

Example: Closed Monoidal Categories

- We can replace certain natural isomorphisms between Hom-sets with adjoint functors and mates.
- A category with both monoidal and closed structures at the same time.

Example: Closed Monoidal Categories

- We can replace certain natural isomorphisms between Hom-sets with adjoint functors and mates.
- A category with both monoidal and closed structures at the same time.
  - Given monoidal functor $- \otimes -$ and inner hom functor $[-,-]$, we have the natural iso:

$$NI : Hom(X \otimes Y, Z) \simeq Hom(X, [Y, Z])$$

Example: Closed Monoidal Categories

- We can replace certain natural isomorphisms between Hom-sets with adjoint functors and mates.
- A category with both monoidal and closed structures at the same time.
  - Given monoidal functor $- \otimes -$ and inner hom functor $[-, -]$, we have the natural iso:

$$NI : Hom(X \otimes Y, Z) \simeq Hom(X, [Y, Z])$$

- Formulate NI using adjunctions and mates:

Example: Closed Monoidal Categories

- We can replace certain natural isomorphisms between Hom-sets with adjoint functors and mates.
- A category with both monoidal and closed structures at the same time.
  - Given monoidal functor $- \otimes -$ and inner hom functor $[-, -]$, we have the natural iso:

$$NI : Hom(X \otimes Y, Z) \simeq Hom(X, [Y, Z])$$

- Formulate NI using adjunctions and mates:
  - Let $L = - \otimes Y$ and $R = [Y, -]$. $L \dashv R \;\Rightarrow\; NI$ natural in X and Z.

Example: Closed Monoidal Categories

- We can replace certain natural isomorphisms between Hom-sets with adjoint functors and mates.
- A category with both monoidal and closed structures at the same time.
  - Given monoidal functor $- \otimes -$ and inner hom functor $[-, -]$, we have the natural iso:

$$NI : Hom(X \otimes Y, Z) \simeq Hom(X, [Y, Z])$$

- Formulate NI using adjunctions and mates:
  - Let $L = - \otimes Y$ and $R = [Y, -]$. $L \dashv R \;\Rightarrow\; NI$ natural in X and Z.
  - For $f : Y \Rightarrow Y'$, $\alpha = - \otimes f$ and $\beta = [f, -]$ form a mate $\Rightarrow NI$ natural in Y.

Example: Closed Monoidal Categories

- We can replace certain natural isomorphisms between Hom-sets with adjoint functors and mates.
- A category with both monoidal and closed structures at the same time.
  - Given monoidal functor $- \otimes -$ and inner hom functor $[-, -]$, we have the natural iso:

$$NI : Hom(X \otimes Y, Z) \simeq Hom(X, [Y, Z])$$

- Formulate NI using adjunctions and mates:
  - Let $L = - \otimes Y$ and $R = [Y, -]$. $L \dashv R \;\Rightarrow\; NI$ natural in X and Z.
  - For $f : Y \Rightarrow Y'$, $\alpha = - \otimes f$ and $\beta = [f, -]$ form a mate $\Rightarrow NI$ natural in Y.
- This definition of closed monoidal categories has no universe level issues

Natural Isomorphisms versus Adjunctions + Mates

- This observation can be generalized.

Natural Isomorphisms versus Adjunctions + Mates

- This observation can be generalized.
- Consider $Hom(L(X, Y_1, \cdots, Y_n), Z) \simeq Hom(X, R(Y_1, \cdots, Y_n, Z))$,

Natural Isomorphisms versus Adjunctions + Mates

- This observation can be generalized.
- Consider $Hom(L(X, Y_1, \cdots, Y_n), Z) \simeq Hom(X, R(Y_1, \cdots, Y_n, Z))$,
  - $L(-, Y_1, \cdots, Y_n) \dashv R(Y_1, \cdots, Y_n, -) \Rightarrow$ naturality in X and Z;

Natural Isomorphisms versus Adjunctions + Mates

- This observation can be generalized.
- Consider $Hom(L(X, Y_1, \cdots, Y_n), Z) \simeq Hom(X, R(Y_1, \cdots, Y_n, Z))$,
  - $L(-, Y_1, \cdots, Y_n) \dashv R(Y_1, \cdots, Y_n, -) \Rightarrow$ naturality in X and Z;
  - for $f_i : Y_i \Rightarrow Y_i', i \in [1, n]$, $\alpha : L(-, f_1, \cdots, f_n)$ and $\beta : R(f_1, \cdots, f_n, -)$ form a mate $\Rightarrow$ naturality in all Y_i.

Natural Isomorphisms versus Adjunctions + Mates

- This observation can be generalized.
- Consider $Hom(L(X, Y_1, \cdots, Y_n), Z) \simeq Hom(X, R(Y_1, \cdots, Y_n, Z))$,
  - $L(-, Y_1, \cdots, Y_n) \dashv R(Y_1, \cdots, Y_n, -) \Rightarrow$ naturality in X and Z;
  - for $f_i : Y_i \Rightarrow Y_i', i \in [1, n]$, $\alpha : L(-, f_1, \cdots, f_n)$ and $\beta : R(f_1, \cdots, f_n, -)$ form a mate $\Rightarrow$ naturality in all Y_i.
  - Using both, regain the naturality of all X, Y_i and Z without using anything set theoretic.

Finiteness of Categories

- Besides smallness and local smallness, we sometimes even require objects (and even morphisms) to be finite.

Finiteness of Categories

- Besides smallness and local smallness, we sometimes even require objects (and even morphisms) to be finite.
- In a set theoretic view,

$$\mathit{Finite}(\mathit{Obj}) \triangleq \sum_{n:\mathbb{N}} \mathit{Bijection}(\mathit{Obj}, \texttt{Fin}\ n)$$

where *Obj* is the type representing objects and *Bijection* is a predicate showing a bijection between *Obj* and $\texttt{Fin}\ n$.

Finiteness of Categories

- Besides smallness and local smallness, we sometimes even require objects (and even morphisms) to be finite.
- In a set theoretic view,

$$Finite(Obj) \triangleq \sum_{n:\mathbb{N}} Bijection(Obj, \texttt{Fin}\ \ n)$$

  where Obj is the type representing objects and $Bijection$ is a predicate showing a bijection between Obj and $\texttt{Fin}\ \ n$.
- There are problems:
  - We are forced to talk about equality between objects.
  - We are implicitly assuming objects are a set.

Adjoint Equivalence

We can use Adjoint Equivalences (A.E.) to quantify finiteness of objects and morphisms.

- A.E. can be used to turn set theoretic constructions into more categorical ones.

Adjoint Equivalence

We can use Adjoint Equivalences (A.E.) to quantify finiteness of objects and morphisms.

- A.E. can be used to turn set theoretic constructions into more categorical ones.

Definition

Two categories $\mathcal{C}$ and $\mathcal{D}$ are adjoint equivalent if there are two functors $L : \mathcal{C} \to \mathcal{D}$ and $R : \mathcal{D} \to \mathcal{C}$ forming an adjunction $L \dashv R$ where the unit and counit are natural isomorphisms.

Finite Diagrams

- We do not want to talk about the finiteness of *any* type as it is not categorical.

Finite Diagrams

- We do not want to talk about the finiteness of *any* type as it is not categorical.
- However, we can do so if a concrete type is given.

Finite Diagrams

- We do not want to talk about the finiteness of *any* type as it is not categorical.
- However, we can do so if a concrete type is given.
- A finite diagram is a special category in which objects and morphisms are finite:

Definition

Given $n : \mathbb{N}$ as the number of objects and a function $|a, b| : \mathbb{N}$ for $a, b : \texttt{Fin n}$, a finite diagram is a category with

1. `Fin n` as objects, and
2. `Fin` $|a, b|$ as morphisms for `a, b : Fin n`.

Finite Categories

- Finite diagrams are a special kind of finite categories.

Finite Categories

- Finite diagrams are a special kind of finite categories.
- Generalize the finiteness of objects and morphisms in finite diagrams by demanding an A.E.

Definition

A category $\mathcal{C}$ is finite, if it is adjoint equivalent to a finite diagram.

Finite Categories

- Finite diagrams are a special kind of finite categories.
- Generalize the finiteness of objects and morphisms in finite diagrams by demanding an A.E.

Definition

A category $\mathcal{C}$ is finite, if it is adjoint equivalent to a finite diagram.

- Well-behaved w.r.t (co)limits:
  - if a finite diagram is the index category of some (co)limit, then adjoint equivalence demonstrates an isomorphic (co)limit with a general finite category as the index.

Finite Categories

- Finite diagrams are a special kind of finite categories.
- Generalize the finiteness of objects and morphisms in finite diagrams by demanding an A.E.

Definition

A category $\mathcal{C}$ is finite, if it is adjoint equivalent to a finite diagram.

- Well-behaved w.r.t (co)limits:
  - if a finite diagram is the index category of some (co)limit, then adjoint equivalence demonstrates an isomorphic (co)limit with a general finite category as the index.
- One could consider other notions of equivalence depending on the purpose.

Conclusion

- The library is available at `https://github.com/agda/agda-categories`!
  - Feel free to submit a PR!

Conclusion

- The library is available at `https://github.com/agda/agda-categories`!
  - Feel free to submit a PR!
- One can use the library to study category theory and related fields.

Conclusion

- The library is available at `https://github.com/agda/agda-categories`!
  - Feel free to submit a PR!
- One can use the library to study category theory and related fields.
- Some formulations are more suitable for type theory.
  - Morphism-level equalities work better than natural isomorphism.
  - Using adjoint equivalence to avoid set theoretic constructs.

Conclusion

- The library is available at https://github.com/agda/agda-categories!
  - Feel free to submit a PR!
- One can use the library to study category theory and related fields.
- Some formulations are more suitable for type theory.
  - Morphism-level equalities work better than natural isomorphism.
  - Using adjoint equivalence to avoid set theoretic constructs.
- Much design space remains to be explored in formalizing category theory!

Conclusion

- The library is available at https://github.com/agda/agda-categories!
  - Feel free to submit a PR!
- One can use the library to study category theory and related fields.
- Some formulations are more suitable for type theory.
  - Morphism-level equalities work better than natural isomorphism.
  - Using adjoint equivalence to avoid set theoretic constructs.
- Much design space remains to be explored in formalizing category theory!
- Check our paper for more discussions!

Bibliography I

Ahrens, B., Kapulkin, K., and Shulman, M. (2015).
Univalent categories and the rezk completion.
Mathematical Structures in Computer Science, 25(5):1010–1039.

Gross, J., Chlipala, A., and Spivak, D. I. (2014).
Experience implementing a performant category-theory library in coq.
In Klein, G. and Gamboa, R., editors, *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8558 of *Lecture Notes in Computer Science*, pages 275–291. Springer.

Huet, G. P. and Saïbi, A. (2000).
Constructive category theory.
In Plotkin, G. D., Stirling, C., and Tofte, M., editors, *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 239–276. The MIT Press.

mathlib Community, T. (2020).
The lean mathematical library.
In Blanchette, J. and Hritcu, C., editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 367–381. ACM.

Peebles, D., Deikun, J., Norell, U., Doel, D., Jahandarie, D., and Cook, J. (2018).
categories: Categories parametrized by morphism equality in agda.

Bibliography II

Stark, E. W. (2016).
Category theory with adjunctions and limits.
Archive of Formal Proofs.
http://isa-afp.org/entries/Category3.html, Formal proof development.

Stark, E. W. (2017).
Monoidal categories.
Archive of Formal Proofs.
http://isa-afp.org/entries/MonoidalCategory.html, Formal proof development.

Stark, E. W. (2020).
Bicategories.
Archive of Formal Proofs.
http://isa-afp.org/entries/Bicategory.html, Formal proof development.

Timany, A. and Jacobs, B. (2016).
Category theory in coq 8.5.
In Kesner, D. and Pientka, B., editors, *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, volume 52 of *LIPIcs*, pages 30:1–30:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Voevodsky, V., Ahrens, B., Grayson, D., et al.
UniMath — a computer-checked library of univalent mathematics.
available at https://github.com/UniMath/UniMath.

Wiegley, J. (2019).
category-theory: Category theory in coq.