What I learned from formalizing Category Theory in Agda

Jacques Carette

McMaster University

Introduction

What I learned formalizing category[1] in Agda:

- To be proficient with, and idiomatic in, Agda,
- Category theory,
- *Lots about the design space.*

Visit https://github.com/agda/agda-categories and submit PRs!

[1] but were rarely new

Design Decisions

Design Decisions:

- Use dependent types “a lot”; stick to standard Agda.

Design Decisions

Design Decisions:

- Use dependent types “a lot”; stick to standard Agda.
- $\approx$ of Hom carries *evidence* and $\circ$ respects it.

Design Decisions

Design Decisions:

- Use dependent types "a lot"; stick to standard Agda.
- $\approx$ of Hom carries *evidence* and $\circ$ respects it.
- Obj $\neq$ Hom $\neq$ evidence of $\approx$.

Design Decisions

Design Decisions:

- Use dependent types “a lot”; stick to standard Agda.
- $\approx$ of Hom carries *evidence* and $\circ$ respects it.
- Obj $\neq$ Hom $\neq$ evidence of $\approx$.
- Obj has no equality.

Design Decisions

Design Decisions:

- Use dependent types "a lot"; stick to standard Agda.
- $\approx$ of Hom carries *evidence* and $\circ$ respects it.
- Obj $\neq$ Hom $\neq$ evidence of $\approx$.
- Obj has no equality.
- Hom is not necessarily a set.

i.e. Setoid-enriched aka E-categories plus universes plus proof-relevance.

Design Decisions

Design Decisions:

- Use dependent types "a lot"; stick to standard Agda.
- $\approx$ of Hom carries *evidence* and $\circ$ respects it.
- Obj $\neq$ Hom $\neq$ evidence of $\approx$.
- Obj has no equality.
- Hom is not necessarily a set.

i.e. Setoid-enriched aka E-categories plus universes plus proof-relevance.

Fewer assumptions *lets you see more*.

What that looks like in Agda

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

    _≈_ : ∀ {A B} → (f g : A ⇒ B) → Set e
    equiv : ∀ {A B} → IsEquivalence (_≈_ {A} {B})
    ∘-resp-≈  : f ≈ h → g ≈ i → f ∘ g ≈ h ∘ i

    -- plus laws
```

op involutive?

Want $(\mathcal{C}^{op})^{op}$ "=" $\mathcal{C}$. (Technically: *definitionally*).

op as a *function* from the *presentation* of a category to another presentation.

op involutive?

Want $(\mathcal{C}^{op})^{op}$ "=" $\mathcal{C}$. (Technically: *definitionally*).

op as a *function* from the *presentation* of a category to another presentation.

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

op involutive?

Want $(\mathcal{C}^{op})^{op}$ "=" $\mathcal{C}$. (Technically: *definitionally*).

op as a *function* from the *presentation* of a category to another presentation.

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

Some concepts, e.g. `Monad` and `NaturalTransformation`, require similar additional laws.

Duals of Constant Functor?

Want a single dual to Functor $F : \top \Rightarrow C$.

Problem: using either left or right identity law to prove that F preserves composition is "wrong".

Duals of Constant Functor?

Want a single dual to Functor $F : \top \Rightarrow C$.

Problem: using either left or right identity law to prove that F preserves composition is "wrong".

```
identity² : ∀ {A} → id ∘ id {A} ≈ id {A}
```

Duals of Constant Functor?

Want a single dual to Functor $F : \top \Rightarrow C$.

Problem: using either left or right identity law to prove that F preserves composition is “wrong”.

```
identity² : ∀ {A} → id ∘ id {A} ≈ id {A}
```

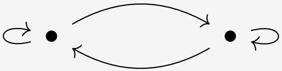
Probably exists a *much better reason* for this, but *I* don't know it!

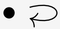
Category of categories exists!

```
Cats : ∀ o ℓ e → Category (suc (o ⊔ ℓ ⊔  e)) (o ⊔ ℓ ⊔ e) (o ⊔ ℓ ⊔ e)
Cats o ℓ e = record
  { Obj        = Category o ℓ e
  ; _⇒_         = Functor
  ; _≈_        = NaturalIsomorphism
  ; id         = id
  ; _∘_        = _∘F_
  ; assoc      = λ {_ _ _ _ F G H} → associator F G H
  ; sym-assoc = λ {_ _ _ _ F G H} → sym (associator F G H)
  ; identityˡ = unitorˡ
  ; identityʳ = unitorʳ
  ; identity² = unitor²
  ; equiv      = isEquivalence
  ; ∘-resp-≈  = _ⓘₕ_
  }
```

Underlying graph, is that a categorical notion?

Consider the following two categories:

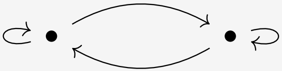




- Are equivalent
- Have different underlying graphs

Underlying graph, is that a categorical notion?

Consider the following two categories:



- Are equivalent
- Have different underlying graphs

```
Underlying : Functor (StrictCats o ℓ e) (Quivers o ℓ e)
PathsOf : Functor (Quivers o ℓ e) (StrictCats o (o ⊔ ℓ) (o ⊔ ℓ ⊔ e))
Free⊣Underlying : Adjoint (PathsOf {o} {o ⊔ ℓ} {o ⊔ ℓ ⊔ e}) Underlying
```

Adjoint Functors: Hom iso?

- 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: Hom iso?

- 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: Hom iso?

- 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: Hom iso?

- 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: Hom iso?

- 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 arises in many definitions / statements involving Homs.

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 hold:

1. $\epsilon L \circ L\eta D. \approx 1_L$ (zig)
2. $R\epsilon \circ \eta RC. \approx 1_R$ (zag)

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 hold:

1. $\epsilon L \circ L\eta D. \approx 1_L$ (zig)
2. $R\epsilon \circ \eta RC. \approx 1_R$ (zag)

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

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 hold:

1. $\epsilon L \circ L \eta D. \approx 1_L$ (zig)
2. $R \epsilon \circ \eta R C. \approx 1_R$ (zag)

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

Fibration?

```
record Fibration {o ℓ e o′ ℓ′ e′} {C : Category o ℓ e} {D : Category o′ ℓ′ e′}
  (F : Functor C D) : Set _ where
  field
    universal₀ : (f : A D.⇒ F₀ B) → C.Obj
    universal₁ : (f : A D.⇒ F₀ B) → universal₀ f C.⇒ B
    iso        : (f : A D.⇒ F₀ B) → F₀ (universal₀ f) ≊ A

  module iso {A B} (f : A D.⇒ F₀ B) = _≊_ (iso f)

  field
    commute   : (f : A D.⇒ F₀ B) → f D.∘ iso.from f D.≈ F₁ (universal₁ f)
    cartesian : (f : A D.⇒ F₀ B) → Cartesian F (universal₁ f)
```

Usability / Engineering lessons: Explicit duals

```
record IsEqualizer {E} (arr : E ⇒ A) (f g : A ⇒ B) : Set _ where
 field
  equality : f ∘ arr ≈ g ∘ arr
  equalize : ∀ {h : X ⇒ A} → f ∘ h ≈ g ∘ h → X ⇒ E
  universal : ∀ {eq : f ∘ h ≈ g ∘ h} → h ≈ arr ∘ equalize eq
  unique    : ∀ {eq : f ∘ h ≈ g ∘ h} → h ≈ arr ∘ i
    → i ≈ equalize eq
record IsCoequalizer {E} (f g : A ⇒ B) (arr : B ⇒ E) : Set _ where
 field
  equality   : arr ∘ f ≈ arr ∘ g
  coequalize : {h : B ⇒ C} → h ∘ f ≈ h ∘ g → E ⇒ C
  universal  : {h : B ⇒ C} {eq : h ∘ f ≈ h ∘ g} → h ≈ coequalize eq ∘ arr
  unique     : {h : B ⇒ C} {i : E ⇒ C} {eq : h ∘ f ≈ h ∘ g} → h ≈ i ∘ arr
    → i ≈ coequalize eq
```

Usability / Engineering lessons: Explicit duals

```
record IsEqualizer {E} (arr : E ⇒ A) (f g : A ⇒ B) : Set _ where
 field
  equality : f ∘ arr ≈ g ∘ arr
  equalize : ∀ {h : X ⇒ A} → f ∘ h ≈ g ∘ h → X ⇒ E
  universal : ∀ {eq : f ∘ h ≈ g ∘ h} → h ≈ arr ∘ equalize eq
  unique    : ∀ {eq : f ∘ h ≈ g ∘ h} → h ≈ arr ∘ i
    → i ≈ equalize eq
```

But it really is more deck chair shuffling:

```
Coequalizer⇔coEqualizer : ∀ (coequalizer : Coequalizer f g) →
 coEqualizer⇒Coequalizer (Coequalizer⇒coEqualizer coequalizer) ≡ coequalizer

Coequalizer⇔coEqualizer _ = refl
```

Usability / Engineering lessons: Predicates vs Structures

```
record IsEqualizer {E} (arr : E ⇒ A) (f g : A ⇒ B) : Set _ where
 field
  equality : f ∘ arr ≈ g ∘ arr
  equalize : ∀ {h : X ⇒ A} → f ∘ h ≈ g ∘ h → X ⇒ E
  universal : ∀ {eq : f ∘ h ≈ g ∘ h} → h ≈ arr ∘ equalize eq
  unique    : ∀ {eq : f ∘ h ≈ g ∘ h} → h ≈ arr ∘ i
    → i ≈ equalize eq

record Equalizer (f g : A ⇒ B) : Set (o ⊔ ℓ ⊔ e) where
 field
  {obj} : Obj
  arr   : obj ⇒ A
  isEqualizer : IsEqualizer arr f g
 open IsEqualizer isEqualizer public
```

Usability / Engineering lessons: Conservative (Definitional) Extensions

```
record IsEqualizer {E} (arr : E ⇒ A) (f g : A ⇒ B) : Set _ where
 field
  equality : f ∘ arr ≈ g ∘ arr
  equalize : ∀ {h : X ⇒ A} → f ∘ h ≈ g ∘ h → X ⇒ E
  universal : ∀ {eq : f ∘ h ≈ g ∘ h} → h ≈ arr ∘ equalize eq
  unique    : ∀ {eq : f ∘ h ≈ g ∘ h} → h ≈ arr ∘ i
    → i ≈ equalize eq

 unique′ : (eq eq′ : f ∘ h ≈ g ∘ h) → equalize eq ≈ equalize eq′
 unique′ eq eq′ = unique universal

 id-equalize : id ≈ equalize equality
 id-equalize = unique (sym identityʳ)

  ...
```

Usability / Engineering lessons: Equational Proofs!

```
record IsEqualizer {E} (arr : E ⇒ A) (f g : A ⇒ B) : Set _ where
 field
  equality : f ∘ arr ≈ g ∘ arr
  equalize : ∀ {h : X ⇒ A} → f ∘ h ≈ g ∘ h → X ⇒ E
  universal : ∀ {eq : f ∘ h ≈ g ∘ h} → h ≈ arr ∘ equalize eq
  unique    : ∀ {eq : f ∘ h ≈ g ∘ h} → h ≈ arr ∘ i
    → i ≈ equalize eq

 equalize-resp-≈ : ∀ {eq : f ∘ h ≈ g ∘ h} {eq′ : f ∘ i ≈ g ∘ i} →
   h ≈ i → equalize eq ≈ equalize eq′
 equalize-resp-≈ {h = h} {i = i} {eq = eq} {eq′ = eq′} h≈i =
   unique $ begin
   i                ≈˘⟨ h≈i ⟩
   h                ≈⟨ universal ⟩
   arr ∘ equalize eq ∎
```

Usability / Engineering lessons: (Un)Bundling

```
open import Categories.Category.Unbundled using (Category)
record IdentityOnObjects {Obj : Set o}
    (C : Category Obj ℓ e) (D : Category Obj ℓ′ e′) : Set _ where
 field
  F₁ : ∀ {A B} → (A C.⇒ B) → A D.⇒ B
  -- laws elided

IOO⇒Functor : {Ob : Set o} {C : Category Ob ℓ e} {D : Category Ob ℓ′ e′} →
  (F : IdentityOnObjects C D) → Functor (pack′ C) (pack′ D)
IOO⇒Functor F = record { F₀ = id→; IOO }
  where module IOO = IdentityOnObjects F
```

Levels as Signals: Comma Category (and thus (co)Slice too)

```
module _ {A : Category o₁ ℓ₁ e₁} {B : Category o₂ ℓ₂ e₂} {C : Category o₃ ℓ₃ e₃
 record CommaObj (T : Functor A C) (S : Functor B C) : Set (o₁ ⊔ o₂ ⊔ ℓ₃) where
  field
   {α} : Obj A
   {β} : Obj B
   f   : C [ T₀ α , S₀ β ]
 record Comma⇒ {T : Functor A C} {S : Functor B C} (X₁ X₂ : CommaObj T S)
   : Set (ℓ₁ ⊔ ℓ₂ ⊔ e₃) where
  field
   g       : A [ α₁ , α₂ ]
   h       : B [ β₁ , β₂ ]
   commute : CommutativeSquare f₁ (T₁ g) (S₁ h) f₂
 Comma : Functor A C → Functor B C
    → Category (o₁ ⊔ o₂ ⊔ ℓ₃) (ℓ₁ ⊔ ℓ₂ ⊔ e₃) (e₁ ⊔ e₂)
```

Levels as Signals: Enriched Functors

```
record Functor (C : Category o ℓ e) (D : Category o′ ℓ′ e′)
    : Set (o ⊔ ℓ ⊔ e ⊔ o′ ⊔ ℓ′ ⊔ e′)

module _
 {o ℓ e} {V : Setoid-Category o ℓ e} (M : Monoidal V) where

  record Category (v : Level) : Set (o ⊔ ℓ ⊔ e ⊔ suc v)
  record Functor {c d} (C : Category c) (D : Category d)
    : Set (ℓ ⊔ e ⊔ c ⊔ d)
```

Maybe "enriched functor" should also do change of base?

Additional bits

More Observations:

- Definitional extensions of Monoidal Category so large that they needed to be split out into own module.
- The category of $\mathrm{Setoids}$ (at a particular level) cannot be a Topos for size/predicatity reasons: the setoid classifier (classifying map) is "too large". (ΠW-Pretopos is ok)
- Multicategory *easier* to do with generalized arities and *relative equations* (implicit combinatorics of $\mathbb{N}$ awful).

Additional bits

More Observations:

- Definitional extensions of Monoidal Category so large that they needed to be split out into own module.
- The category of $\mathrm{Setoids}$ (at a particular level) cannot be a Topos for size/predicatity reasons: the setoid classifier (classifying map) is "too large". (ΠW-Pretopos is ok)
- Multicategory *easier* to do with generalized arities and *relative equations* (implicit combinatorics of $\mathbb{N}$ awful).

Conjecture 1: The Category of -1-Categories, seen as the collection of Enriched categories over the Monoidal -2-Category, is equivalent to the Category $\mathbb{2}$, is equivalent to Excluded Middle.

Additional bits

More Observations:

- Definitional extensions of Monoidal Category so large that they needed to be split out into own module.
- The category of $\mathrm{Setoids}$ (at a particular level) cannot be a Topos for size/predicatity reasons: the setoid classifier (classifying map) is "too large". (ΠW-Pretopos is ok)
- Multicategory *easier* to do with generalized arities and *relative equations* (implicit combinatorics of $\mathbb{N}$ awful).

Conjecture 1: The Category of -1-Categories, seen as the collection of Enriched categories over the Monoidal -2-Category, is equivalent to the Category $\mathbb{2}$, is equivalent to Excluded Middle.

Conjecture 2: Discr is not left adjoint of the forgetful Functor from Cats to $\mathrm{Setoids}$.

Conclusion

- CT in Agda, Lean, cubical Agda, Coq, Coq/HoTT, Isabelle, . . .
  $\Rightarrow$ Category Theory is robust wrt foundations
- Setoid-enriched weak Category Theory is akin "1.5" Category Theory
- Tremendous fun!