The Yoneda Lemma

Siddharth Bhat

##harmless Category Theory in Context

July 25 2021

The statement

Given

- A locally small category C
- A functor $F: C \rightarrow Set$
- An element $a \in C$

Question: How many natural transformations $\eta: Hom(a, -) \Rightarrow F$ are

there?

Answer: Exactly as many as |F(a)| (where

 $F(a) \in Obj(Set)$)

How? Establish a bijection between elements $x \in F(a)$ and natural

transformations $\eta_x : Hom(a, -) \Rightarrow F$.

```
type Hom\ a\ b = a \rightarrow b
type Reader a = Hom a
type Nat f g = forall x. f x \rightarrow g x
```

```
yoFwd :: Nat (Hom a) f -> f a
yoBwd :: f a -> Nat (Hom a) f
```

```
type Hom a b = a -> b
type Reader a = Hom a
type Nat f g = forall x. f x -> g x

yoFwd :: Nat (Hom a) f -> f a
yoFwd :: (forall x. (Hom a x) -> f x) -> f a
yoFwd :: (forall x. (a -> x) -> f x) -> f a
yoFwd ak = (ak @ a) (id @ a:: a -> a) :: f a
-- ak @ a :: (a -> a) -> f a; id @ a :: a -> a

yoBwd :: f a -> Nat (Hom a) f
```

■ Given the element $a \in C$, functor $F : C \to \text{Set}$, natural transformation $\eta : Hom(a, -) \Rightarrow F$, we must produce an element of F(a).

```
type Hom a b = a -> b

type Reader a = Hom a

type Nat f g = forall x. f x -> g x

yoFwd :: Nat (Hom a) f -> f a

yoFwd :: (forall x. (Hom a x) -> f x) -> f a

yoFwd :: (forall x. (a -> x) -> f x) -> f a

yoFwd ak = (ak @ a) (id @ a:: a -> a) :: f a

-- ak @ a :: (a -> a) -> f a; id @ a :: a -> a

yoBwd :: f a -> Nat (Hom a) f

Given the element a \in C, functor F : C \rightarrow Set, natural transformation
```

- $\eta: Hom(a, -) \Rightarrow F$, we must produce an element of F(a).
- Idea: see that $id_a \in Hom(a, a)$.

```
type Hom a b = a -> b
type Reader a = Hom a
type Nat f g = forall x. f x -> g x

yoFwd :: Nat (Hom a) f -> f a
yoFwd :: (forall x. (Hom a x) -> f x) -> f a
yoFwd :: (forall x. (a -> x) -> f x) -> f a
yoFwd ak = (ak @ a) (id @ a:: a -> a) :: f a
-- ak @ a :: (a -> a) -> f a; id @ a :: a -> a
yoBwd :: f a -> Nat (Hom a) f
```

- Given the element $a \in C$, functor $F : C \to \text{Set}$, natural transformation $\eta : Hom(a, -) \Rightarrow F$, we must produce an element of F(a).
- Idea: see that $id_a \in Hom(a, a)$.
- See that the natural transformation at point a is $\eta_a : Hom(a, a) \Rightarrow Fa$.

```
type Hom a b = a -> b
type Reader a = Hom a
type Nat f g = forall x. f x -> g x

yoFwd :: Nat (Hom a) f -> f a
yoFwd :: (forall x. (Hom a x) -> f x) -> f a
yoFwd :: (forall x. (a -> x) -> f x) -> f a
yoFwd ak = (ak @ a) (id @ a:: a -> a) :: f a
-- ak @ a :: (a -> a) -> f a; id @ a :: a -> a
yoBwd :: f a -> Nat (Hom a) f
```

- Given the element $a \in C$, functor $F : C \to \text{Set}$, natural transformation $\eta : Hom(a, -) \Rightarrow F$, we must produce an element of F(a).
- Idea: see that $id_a \in Hom(a, a)$.
- See that the natural transformation at point a is $\eta_a : Hom(a, a) \Rightarrow Fa$.
- Combine the two, apply $\eta_a(id_a)$: Fa.

```
type Hom a b = a -> b
type Reader a = Hom a
type Nat f g = forall x. f x -> g x

yoFwd :: Nat (Hom a) f -> f a
yoFwd :: (forall x. (Hom a x) -> f x) -> f a
yoFwd :: (forall x. (a -> x) -> f x) -> f a
yoFwd ak = (ak @ a) (id @ a:: a -> a) :: f a
-- ak @ a :: (a -> a) -> f a; id @ a :: a -> a
yoBwd :: f a -> Nat (Hom a) f
```

- Given the element $a \in C$, functor $F : C \to \text{Set}$, natural transformation $\eta : Hom(a, -) \Rightarrow F$, we must produce an element of F(a).
- Idea: see that $id_a \in Hom(a, a)$.
- See that the natural transformation at point a is $\eta_a : Hom(a, a) \Rightarrow Fa$.
- Combine the two, apply $\eta_a(id_a)$: Fa.

```
type Hom a b = a -> b; type Reader a = Hom a; type Nat f g = forall x. f x -> g x yoBwd :: f a -> Nat (Hom a) f yoBwd :: f a -> (forall x. (a -> x) -> f x) yoBwd :: forall x. f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x
```

```
yoFwd :: Nat (Hom a) f -> f a
yoFwd ak = (ak @ a) (id @ a:: a -> a) :: f a
```

■ Given the element $a \in C$, functor $F : C \to \text{Set}$, an element $fa \in F(a)$, we must produce a natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F$.

```
type Hom a b = a -> b; type Reader a = Hom a; type Nat f g = forall x. f x -> g x yoBwd :: f a -> Nat (Hom a) f yoBwd :: f a -> (forall x. (a -> x) -> f x) yoBwd :: forall x. f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x
```

```
yoFwd :: Nat (Hom a) f -> f a
yoFwd ak = (ak @ a) (id @ a:: a -> a) :: f a
```

- Given the element $a \in C$, functor $F : C \to \text{Set}$, an element $fa \in F(a)$, we must produce a natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F$.
- For each $fa \in F(a)$, let's define the natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F .$

```
type Hom a b = a -> b; type Reader a = Hom a; type Nat f g = forall x. f x -> g x yoBwd :: f a -> Nat (Hom a) f yoBwd :: f a -> (forall x. (a -> x) -> f x) yoBwd :: forall x. f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x
```

```
yoFwd :: Nat (Hom a) f -> f a
yoFwd ak = (ak @ a) (id @ a:: a -> a) :: f a
```

- Given the element $a \in C$, functor $F : C \to \text{Set}$, an element $fa \in F(a)$, we must produce a natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F$.
- For each $fa \in F(a)$, let's define the natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F-$.
- Define it component-wise. For each $x \in C$, consider $\eta_{fa}(x) : Hom(a, -)(x) \Rightarrow F(x)$.

```
type Hom a b = a -> b; type Reader a = Hom a; type Nat f g = forall x. f x -> g x yoBwd :: f a -> Nat (Hom a) f yoBwd :: f a -> (forall x. (a -> x) -> f x) yoBwd :: forall x. f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x
```

```
yoFwd :: Nat (Hom a) f -> f a
yoFwd ak = (ak @ a) (id @ a:: a -> a) :: f a
```

- Given the element $a \in C$, functor $F : C \to \text{Set}$, an element $fa \in F(a)$, we must produce a natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F$.
- For each $fa \in F(a)$, let's define the natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F-$.
- Define it component-wise. For each $x \in C$, consider $\eta_{fa}(x) : Hom(a, -)(x) \Rightarrow F(x)$.
- Take an arrow $a \xrightarrow{aUser} x \in Hom(a, -)(x)$.

```
type Hom a b = a -> b; type Reader a = Hom a; type Nat f g = forall x. f x -> g x yoBwd :: f a -> Nat (Hom a) f yoBwd :: f a -> (forall x. (a -> x) -> f x) yoBwd :: forall x. f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x
```

```
yoFwd :: Nat (Hom a) f -> f a
yoFwd ak = (ak @ a) (id @ a:: a -> a) :: f a
```

- Given the element $a \in C$, functor $F : C \to \text{Set}$, an element $fa \in F(a)$, we must produce a natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F$.
- For each $fa \in F(a)$, let's define the natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F-$.
- Define it component-wise. For each $x \in C$, consider $\eta_{fa}(x) : Hom(a, -)(x) \Rightarrow F(x)$.
- Take an arrow $a \xrightarrow{aUser} x \in Hom(a, -)(x)$.
- Send it to $F(a) \xrightarrow{F(aUser)} F(x)$.

```
type Hom a b = a -> b; type Reader a = Hom a; type Nat f g = forall x. f x -> g x

yoBwd :: f a -> Nat (Hom a) f
yoBwd :: f a -> (forall x. (a -> x) -> f x)
yoBwd :: forall x. f a -> (a -> x) -> f x
yoBwd :: f a -> (a -> x) -> f x
yoBwd :: f a -> (a -> x) -> f x
```

```
yoFwd :: Nat (Hom a) f -> f a
yoFwd ak = (ak @ a) (id @ a:: a -> a) :: f a
```

- Given the element $a \in C$, functor $F : C \to \text{Set}$, an element $fa \in F(a)$, we must produce a natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F$.
- For each $fa \in F(a)$, let's define the natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F-$.
- Define it component-wise. For each $x \in C$, consider $\eta_{fa}(x) : Hom(a, -)(x) \Rightarrow F(x)$.
- Take an arrow $a \xrightarrow{aUser} x \in Hom(a, -)(x)$.
- Send it to $F(a) \xrightarrow{F(aUser)} F(x)$.
- Apply it on $fa \in F(a)$, giving $F(aUser)(fa) \in F(x)$.

```
type Hom a b = a -> b; type Reader a = Hom a; type Nat f g = forall x. f x -> g x yoBwd :: f a -> Nat (Hom a) f yoBwd :: f a -> (forall x. (a -> x) -> f x) yoBwd :: forall x. f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x
```

```
yoFwd :: Nat (Hom a) f -> f a
yoFwd ak = (ak @ a) (id @ a:: a -> a) :: f a
```

- Given the element $a \in C$, functor $F : C \to \text{Set}$, an element $fa \in F(a)$, we must produce a natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F$.
- For each $fa \in F(a)$, let's define the natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F-$.
- Define it component-wise. For each $x \in C$, consider $\eta_{fa}(x) : Hom(a, -)(x) \Rightarrow F(x)$.
- Take an arrow $a \xrightarrow{aUser} x \in Hom(a, -)(x)$.
- Send it to $F(a) \xrightarrow{F(aUser)} F(x)$.
- Apply it on $fa \in F(a)$, giving $F(aUser)(fa) \in F(x)$.
- Success! elements $fa \in F(a)$ are natural transformations η_{fa} defined componentwise.

```
type Hom a b = a -> b; type Reader a = Hom a; type Nat f g = forall x. f x -> g x yoBwd :: f a -> Nat (Hom a) f yoBwd :: f a -> (forall x. (a -> x) -> f x) yoBwd :: forall x. f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x
```

```
yoFwd :: Nat (Hom a) f -> f a
yoFwd ak = (ak @ a) (id @ a:: a -> a) :: f a
```

- Given the element $a \in C$, functor $F: C \to \mathsf{Set}$, an element $fa \in F(a)$, we must produce a natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F$.
- For each $fa \in F(a)$, let's define the natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F-$.
- Define it component-wise. For each $x \in C$, consider $\eta_{fa}(x) : Hom(a, -)(x) \Rightarrow F(x)$.
- Take an arrow $a \xrightarrow{aUser} x \in Hom(a, -)(x)$.
- Send it to $F(a) \xrightarrow{F(aUser)} F(x)$.
- Apply it on $fa \in F(a)$, giving $F(aUser)(fa) \in F(x)$.
- Success! elements $f_a \in F(a)$ are natural transformations η_{f_a} defined componentwise.
- Naturality?

```
type Hom a b = a -> b; type Reader a = Hom a; type Nat f g = forall x. f x -> g x yoBwd :: f a -> Nat (Hom a) f yoBwd :: f a -> (forall x. (a -> x) -> f x) yoBwd :: forall x. f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x yoBwd :: f a -> (a -> x) -> f x
```

```
yoFwd :: Nat (Hom a) f -> f a
yoFwd ak = (ak @ a) (id @ a:: a -> a) :: f a
```

- Given the element $a \in C$, functor $F: C \to \mathsf{Set}$, an element $fa \in F(a)$, we must produce a natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F$.
- For each $fa \in F(a)$, let's define the natural transformation $\eta_{fa} : Hom(a, -) \Rightarrow F-$.
- Define it component-wise. For each $x \in C$, consider $\eta_{fa}(x) : Hom(a, -)(x) \Rightarrow F(x)$.
- Take an arrow $a \xrightarrow{aUser} x \in Hom(a, -)(x)$.
- Send it to $F(a) \xrightarrow{F(aUser)} F(x)$.
- Apply it on $fa \in F(a)$, giving $F(aUser)(fa) \in F(x)$.
- Success! elements $f_a \in F(a)$ are natural transformations η_{f_a} defined componentwise.
- Naturality?

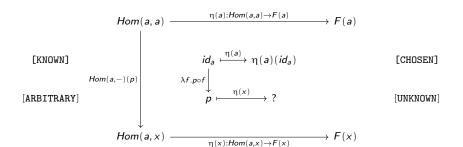
$$id_a: Hom(a, a) \xrightarrow{\eta} \eta(id_a): F(a)$$

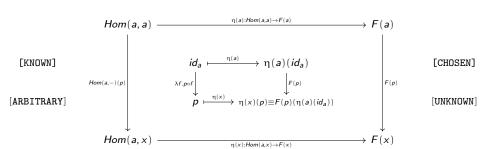
[ARBITRARY]
$$p \in Hom(a, x) \xrightarrow{\eta} \eta(p) = ? : F(x)$$

$$Hom(a,a) \longrightarrow \stackrel{\eta(a):Hom(a,a) \to F(a)}{\longrightarrow} F(a)$$

$$id_a \stackrel{\eta(a)}{\longmapsto} \eta(a)(id_a) \qquad \qquad \text{[CHOSEN]}$$

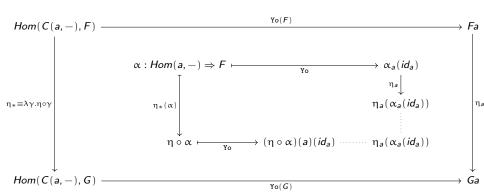
[KNOWN] $id_a \xrightarrow{\eta(a)} \eta(a)(id_a)$ [CHOSEN]





Naturality in the functor

Given a natural transformation $\eta: F \Rightarrow G$, we wish to show the diagram commutes:



Naturality in the object

Given an arrow $a \stackrel{f}{\rightarrow} b$, in C, we wish to show that this diagram commutes:

