The Yoneda Lemma

Siddharth Bhat

##harmless **Category Theory in Context**

July 25 2021

The statement

Given

- A locally small category C
- A functor $F : C \to \mathrm{Set}$
- 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 $F(a) \in Obj(\mathrm{Set})$)
How? Establish a bijection between elements $x \in F(a)$ and natural transformations $\eta_x : Hom(a, -) \Rightarrow F$.

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

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

The proof (1) (natural transformations $\eta : Hom(a, -) \Rightarrow F$ have 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 $F : C \to \mathrm{Set}$, natural transformation $\eta : Hom(a, -) \Rightarrow F$, we must produce an element of $F(a)$.

The proof (1) (natural transformations $\eta : Hom(a, -) \Rightarrow F$ have 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 $F : C \to \text{Set}$, natural transformation $\eta : Hom(a, -) \Rightarrow F$, we must produce an element of $F(a)$.
- Idea: see that $id_a \in Hom(a, a)$.

The proof (1) (natural transformations $\eta : Hom(a, -) \Rightarrow F$ have an element of $F(a)$)

```haskell
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 $F : C \to \mathrm{Set}$, 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 $\eta_a : Hom(a, a) \Rightarrow Fa$.

The proof (1) (natural transformations $\eta : Hom(a, -) \Rightarrow F$ have an element of $F(a)$)

```haskell
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 $\eta_a(id_a) : Fa$.

The proof (1) (natural transformations $\eta : Hom(a, -) \Rightarrow F$ have 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 $F : C \to \text{Set}$, 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 $\eta_a(id_a) : Fa$.

The proof (2) (an element $x \in F(a)$ creates a natural transformation $\eta_a : Hom(a, -) \Rightarrow F$)

```
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 fa aUser = (fmap @ f) aUser fa

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 $\eta_{fa} : Hom(a, -) \Rightarrow F$.

The proof (2) (an element $x \in F(a)$ creates a natural transformation $\eta_a : Hom(a,-) \Rightarrow F$)

```
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 fa aUser = (fmap @ f) aUser fa

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

- Given the element $a \in C$, functor $F : C \to \mathrm{Set}$, 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 $\eta_{fa} : Hom(a,-) \Rightarrow F-$.

The proof (2) (an element $x \in F(a)$ creates a natural transformation $\eta_a : Hom(a, -) \Rightarrow F$)

```
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 fa aUser = (fmap @ f) aUser fa

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 $\eta_{fa}(x) : Hom(a, -)(x) \Rightarrow F(x)$.

The proof (2) (an element $x \in F(a)$ creates a natural transformation $\eta_a : Hom(a, -) \Rightarrow F$)

```
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 fa aUser = (fmap @ f) aUser fa

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 $a \xrightarrow{aUser} x \in Hom(a, -)(x)$.

The proof (2) (an element $x \in F(a)$ creates a natural transformation $\eta_a : Hom(a, -) \Rightarrow F$)

```
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 fa aUser = (fmap @ f) aUser fa

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 $F(a) \xrightarrow{F(aUser)} F(x)$.

The proof (2) (an element $x \in F(a)$ creates a natural transformation $\eta_a : Hom(a, -) \Rightarrow F$)

```
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 fa aUser = (fmap @ f) aUser fa

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 $F(aUser)(fa) \in F(x)$.

The proof (2) (an element $x \in F(a)$ creates a natural transformation $\eta_a : Hom(a, -) \Rightarrow F$)

```
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 fa aUser = (fmap @ f) aUser fa

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.

The proof (2) (an element $x \in F(a)$ creates a natural transformation $\eta_a : Hom(a, -) \Rightarrow F$)

```
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 fa aUser = (fmap @ f) aUser fa

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.
- Naturality?

The proof (2) (an element $x \in F(a)$ creates a natural transformation $\eta_a : Hom(a, -) \Rightarrow F$)

```
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 fa aUser = (fmap @ f) aUser fa

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.
- Naturality?

Proof from the book

[KNOWN] $id_a : Hom(a,a) \xrightarrow{\eta} \eta(id_a) : F(a)$ [CHOSEN]

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

Proof from the book

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

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

Proof from the book

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

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

[ARBITRARY] $\quad p \xmapsto{\eta(p)} ?$ $\quad$ [UNKNOWN]

$$Hom(a,x) \xrightarrow[\eta(x):Hom(a,x)\to F(x)]{} F(x)$$

Proof from the book

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

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

[ARBITRARY] $\quad p \xmapsto{\eta(x)} ? \quad$ [UNKNOWN]

$$Hom(a,x) \xrightarrow[\eta(x):Hom(a,x)\to F(x)]{} F(x)$$

Proof from the book

$$
\begin{tikzcd}
 & Hom(a,a) \arrow[rrr, "\eta(a):Hom(a,a)\to F(a)"] \arrow[ddd, "Hom(a,-)(p)"'] & & & F(a) & \\
\text{[KNOWN]} & & id_a \arrow[r, mapsto, "\eta(a)"] \arrow[d, "\lambda f.p\circ f"'] & \eta(a)(id_a) & & \text{[CHOSEN]} \\
\text{[ARBITRARY]} & & p \arrow[r, mapsto, "\eta(x)"] & ? & & \text{[UNKNOWN]} \\
 & Hom(a,x) \arrow[rrr, "\eta(x):Hom(a,x)\to F(x)"'] & & & F(x) &
\end{tikzcd}
$$

Proof from the book

$$
\begin{tikzcd}
 & Hom(a,a) \arrow[rrr, "\eta(a):Hom(a,a)\to F(a)"] \arrow[ddd, "Hom(a,-)(p)"'] & & & F(a) \arrow[ddd, "F(p)"] & \\
\text{[KNOWN]} & & id_a \arrow[r, mapsto, "\eta(a)"] \arrow[d, "\lambda f.p\circ f"'] & \eta(a)(id_a) \arrow[d, "F(p)"] & & \text{[CHOSEN]} \\
\text{[ARBITRARY]} & & p \arrow[r, mapsto, "\eta(x)"] & \eta(x)(p)\equiv F(p)(\eta(a)(id_a)) & & \text{[UNKNOWN]} \\
 & Hom(a,x) \arrow[rrr, "\eta(x):Hom(a,x)\to F(x)"'] & & & F(x) &
\end{tikzcd}
$$