

$$\begin{array}{ccc}
 \text{Hom}(A, A) & \xrightarrow{\text{Hom}(A, f)} & \text{Hom}(A, X) \\
 \downarrow \Phi_A & & \downarrow \Phi_X \\
 F(A) & \xrightarrow{Ff} & F(X)
 \end{array}$$

$\begin{array}{ccc} \text{id}_A & \xrightarrow{\quad} & f \\ \downarrow & & \downarrow \\ u & \xrightarrow{\quad} & (Ff)u = \Phi_X(f) \end{array}$

## What you needa know about Yoneda

Emma Bach (she/her)

Seminar on Functional Programming and Logic, Summer Semester 2025

# Motivation

- ▶ A common sentiment in many cultures is the idea that things are defined by how they interact with their surroundings.

---

<sup>1</sup>Quoted as a proverb in *Don Quixote*

# Motivation

- ▶ A common sentiment in many cultures is the idea that things are defined by how they interact with their surroundings.
- ▶ “*Tell me your company, and I will tell you what you are.*”<sup>1</sup>

---

<sup>1</sup>Quoted as a proverb in *Don Quixote*

# Motivation

- ▶ A common sentiment in many cultures is the idea that things are defined by how they interact with their surroundings.
- ▶ “*Tell me your company, and I will tell you what you are.*”<sup>1</sup>
- ▶ The Yoneda lemma is the result of applying this way of thinking to mathematical objects in category theory.

---

<sup>1</sup>Quoted as a proverb in *Don Quixote*

# Motivation

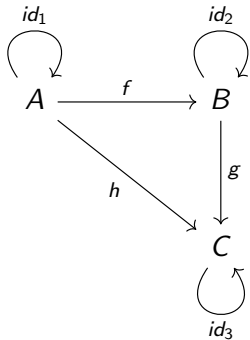
- ▶ A common sentiment in many cultures is the idea that things are defined by how they interact with their surroundings.
- ▶ “*Tell me your company, and I will tell you what you are.*”<sup>1</sup>
- ▶ The Yoneda lemma is the result of applying this way of thinking to mathematical objects in category theory.
- ▶ As a result, a category  $\mathbb{C}$  is often best understood by instead studying functors from that category into  $\mathbf{Set}$ .

---

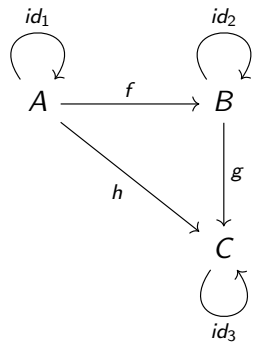
<sup>1</sup>Quoted as a proverb in *Don Quixote*

# Categories

- A category  $\mathbb{C}$  consists of:

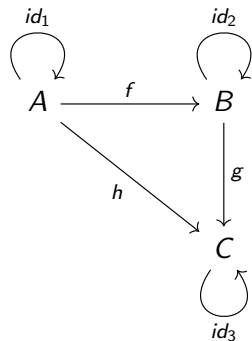


# Categories



- ▶ A *category*  $\mathbb{C}$  consists of:
  - ▶ a collection  $|\mathbb{C}|$  of *objects*;

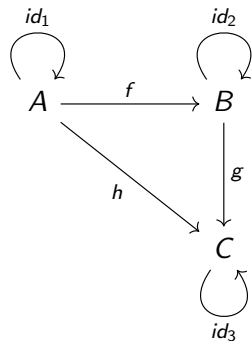
# Categories



- ▶ A *category*  $\mathbb{C}$  consists of:
  - ▶ a collection  $|\mathbb{C}|$  of *objects*;
  - ▶ for all  $A, B \in |\mathbb{C}|$ , a collection  $\mathbb{C}(A, B)$  of *morphisms* from  $A$  to  $B$ , if this is a set we call it a *homset*;

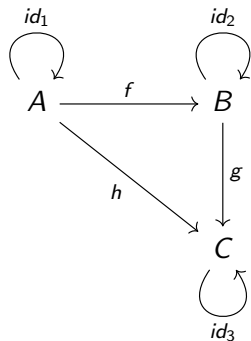


# Categories



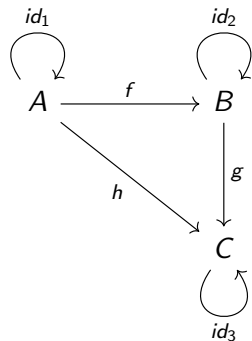
- ▶ A *category*  $\mathbb{C}$  consists of:
  - ▶ a collection  $|\mathbb{C}|$  of *objects*;
  - ▶ for all  $A, B \in |\mathbb{C}|$ , a collection  $\mathbb{C}(A, B)$  of *morphisms* from  $A$  to  $B$ , if this is a set we call it a *homset*;
  - ▶ for all  $A \in |\mathbb{C}|$ , an *identity morphism*  $id_A \in \mathbb{C}(A, A)$ ;

# Categories



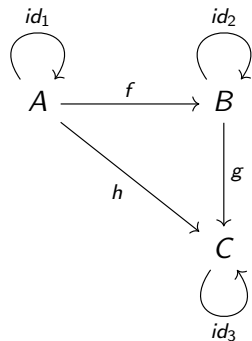
- ▶ A *category*  $\mathbb{C}$  consists of:
  - ▶ a collection  $|\mathbb{C}|$  of *objects*;
  - ▶ for all  $A, B \in |\mathbb{C}|$ , a collection  $\mathbb{C}(A, B)$  of *morphisms* from  $A$  to  $B$ , if this is a set we call it a *homset*;
  - ▶ for all  $A \in |\mathbb{C}|$ , an *identity morphism*  $id_A \in \mathbb{C}(A, A)$ ;
  - ▶ for each pair of morphisms  $g \in \mathbb{C}(B, C)$ ,  $f \in \mathbb{C}(A, B)$ , a morphism  $g \circ f \in \mathbb{C}(A, C)$ , such that composition is associative.

# Categories



- ▶ A *category*  $\mathbb{C}$  consists of:
  - ▶ a collection  $|\mathbb{C}|$  of *objects*;
  - ▶ for all  $A, B \in |\mathbb{C}|$ , a collection  $\mathbb{C}(A, B)$  of *morphisms* from  $A$  to  $B$ , if this is a set we call it a *homset*;
  - ▶ for all  $A \in |\mathbb{C}|$ , an *identity morphism*  $id_A \in \mathbb{C}(A, A)$ ;
  - ▶ for each pair of morphisms  $g \in \mathbb{C}(B, C)$ ,  $f \in \mathbb{C}(A, B)$ , a morphism  $g \circ f \in \mathbb{C}(A, C)$ , such that composition is associative.
- ▶ For every category  $\mathbb{C}$ , the *opposite category*  $\mathbb{C}^{op}$  is a category.

# Categories



- ▶ A *category*  $\mathbb{C}$  consists of:
  - ▶ a collection  $|\mathbb{C}|$  of *objects*;
  - ▶ for all  $A, B \in |\mathbb{C}|$ , a collection  $\mathbb{C}(A, B)$  of *morphisms* from  $A$  to  $B$ , if this is a set we call it a *homset*;
  - ▶ for all  $A \in |\mathbb{C}|$ , an *identity morphism*  $id_A \in \mathbb{C}(A, A)$ ;
  - ▶ for each pair of morphisms  $g \in \mathbb{C}(B, C)$ ,  $f \in \mathbb{C}(A, B)$ , a morphism  $g \circ f \in \mathbb{C}(A, C)$ , such that composition is associative.
- ▶ For every category  $\mathbb{C}$ , the *opposite category*  $\mathbb{C}^{op}$  is a category.
- ▶ For every pair of categories  $\mathbb{C}, \mathbb{D}$ , the *product category*  $\mathbb{C} \times \mathbb{D}$  is a category.

# Functors

A *functor*  $F : \mathbb{C} \rightarrow \mathbb{D}$  is a structure-preserving map between two categories:

$$\begin{array}{ccc} A & \xrightarrow{f \in \mathbb{C}(A,B)} & B \\ \vdots F \downarrow & & \downarrow \vdots F \\ F(A) & \xrightarrow{F(f)} & F(B) \end{array}$$

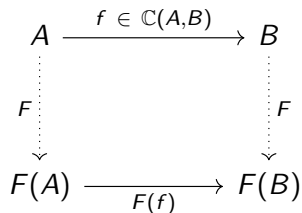
# Functors

A *functor*  $F : \mathbb{C} \rightarrow \mathbb{D}$  is a structure-preserving map between two categories:

- ▶  $F$  maps an object  $A \in |\mathbb{C}|$  to an object  $F(A) \in |\mathbb{D}|$

$$\begin{array}{ccc} A & \xrightarrow{f \in \mathbb{C}(A,B)} & B \\ \text{\scriptsize $F$} \downarrow \text{\scriptsize $\vee$} & & \downarrow \text{\scriptsize $F$} \text{\scriptsize $\vee$} \\ F(A) & \xrightarrow{F(f)} & F(B) \end{array}$$

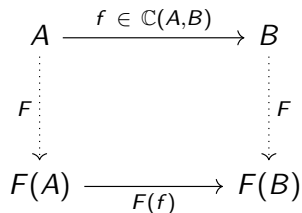
# Functors



A *functor*  $F : \mathbb{C} \rightarrow \mathbb{D}$  is a structure-preserving map between two categories:

- ▶  $F$  maps an object  $A \in |\mathbb{C}|$  to an object  $F(A) \in |\mathbb{D}|$
- ▶  $F$  maps a morphism  $f \in \mathbb{C}(A, B)$  to a morphism  $F(f) \in \mathbb{D}(F(A), F(B))$

# Functors

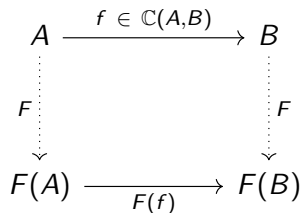


A *functor*  $F : \mathbb{C} \rightarrow \mathbb{D}$  is a structure-preserving map between two categories:

- ▶  $F$  maps an object  $A \in |\mathbb{C}|$  to an object  $F(A) \in |\mathbb{D}|$
- ▶  $F$  maps a morphism  $f \in \mathbb{C}(A, B)$  to a morphism  $F(f) \in \mathbb{D}(F(A), F(B))$
- ▶  $F(id_A) = id_{F(A)}$



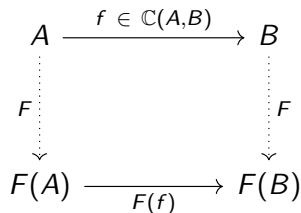
# Functors



A *functor*  $F : \mathbb{C} \rightarrow \mathbb{D}$  is a structure-preserving map between two categories:

- ▶  $F$  maps an object  $A \in |\mathbb{C}|$  to an object  $F(A) \in |\mathbb{D}|$
- ▶  $F$  maps a morphism  $f \in \mathbb{C}(A, B)$  to a morphism  $F(f) \in \mathbb{D}(F(A), F(B))$
- ▶  $F(id_A) = id_{F(A)}$
- ▶  $F(g \circ f) = F(g) \circ F(f)$

# Functors



A *functor*  $F : \mathbb{C} \rightarrow \mathbb{D}$  is a structure-preserving map between two categories:

- ▶  $F$  maps an object  $A \in |\mathbb{C}|$  to an object  $F(A) \in |\mathbb{D}|$
- ▶  $F$  maps a morphism  $f \in \mathbb{C}(A, B)$  to a morphism  $F(f) \in \mathbb{D}(F(A), F(B))$
- ▶  $F(id_A) = id_{F(A)}$
- ▶  $F(g \circ f) = F(g) \circ F(f)$

Functors from a category into itself are known as *endofunctors*.

# Natural Transformations

$$A \xrightarrow{f} B$$

► Structure-preserving maps between functors.

$$\begin{array}{ccc} F(A) & \xrightarrow{F(f)} & F(B) \\ \downarrow \phi_A & & \downarrow \phi_B \\ G(A) & \xrightarrow{G(f)} & G(B) \end{array}$$

# Natural Transformations

$$A \xrightarrow{f} B$$

$$\begin{array}{ccc} F(A) & \xrightarrow{F(f)} & F(B) \\ \downarrow \phi_A & & \downarrow \phi_B \\ G(A) & \xrightarrow{G(f)} & G(B) \end{array}$$

- ▶ Structure-preserving maps between functors.
  - ▶ Let  $F, G : \mathbb{C} \rightarrow \mathbb{D}$  be functors.

# Natural Transformations

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ F(A) & \xrightarrow{F(f)} & F(B) \\ \downarrow \phi_A & & \downarrow \phi_B \\ G(A) & \xrightarrow{G(f)} & G(B) \end{array}$$

- ▶ Structure-preserving maps between functors.
  - ▶ Let  $F, G : \mathbb{C} \rightarrow \mathbb{D}$  be functors.
  - ▶ A natural transformation  $\phi$  is an *indexed family of morphisms* - for every object  $A \in |\mathbb{C}|$ ,  $\phi_A$  is a morphism from  $F(A)$  to  $G(A)$ .

# Natural Transformations

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ \\ F(A) & \xrightarrow{F(f)} & F(B) \\ \downarrow \phi_A & & \downarrow \phi_B \\ G(A) & \xrightarrow{G(f)} & G(B) \end{array}$$

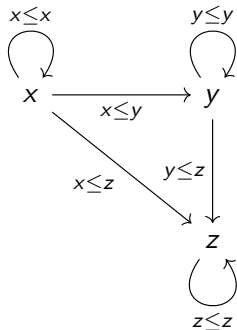
- ▶ Structure-preserving maps between functors.
  - ▶ Let  $F, G : \mathbb{C} \rightarrow \mathbb{D}$  be functors.
  - ▶ A natural transformation  $\phi$  is an *indexed family of morphisms* - for every object  $A \in |\mathbb{C}|$ ,  $\phi_A$  is a morphism from  $F(A)$  to  $G(A)$ .
  - ▶ These morphisms satisfy the *naturality condition*:

$$\forall f \in \mathbb{C}(A, B) : \phi_B \circ F(f) = G(f) \circ \phi_A$$

## Exercise 1 : Order Categories

- a) Let  $\leq$  be a reflexive, transitive order (a *preorder*) on a set  $M$ . Show that if we define objects by  $|\mathbb{P}re(M, \leq)| = M$  and morphisms by  $\exists! f_{x \leq y} \in \mathbb{P}re(x, y) \Leftrightarrow x \leq y$ , then  $\mathbb{P}re(M, \leq)$  forms a category.
- b) Let  $F : \mathbb{M} \rightarrow \mathbb{M}$  be an endofunctor on  $\mathbb{M}$ . Show that  $F$  defines a monotonic function  $M \rightarrow M$ , i.e.  $\forall x, y : x \leq y \implies F(x) \leq F(y)$ .
- c) Let  $F, G : M \rightarrow M$  be monotonic functions. Let  $\phi$  be a natural transformation  $F \rightarrow G$ . Show that  $\forall x \in M : F(x) \leq G(x)$ .

## Exercise 1 : Order Categories, Solution a)



- ▶  $\leq$  is reflexive, so we have  
 $\forall x : x \leq x \implies \exists id_{x \leq x} \in \mathbb{P}re(x, x).$
- ▶ Because of transitivity, for every pair of morphisms  $f_{x \leq y}$  and  $g_{y \leq z}$ , we have a composed morphism  $(g \circ f)_{x \leq z}$ .
- ▶ Since our morphisms are just witnesses of an ordering, they don't care about the order of function application, so composition is associative.



## Exercise 1 : Order Categories, Solution b)

$$\begin{array}{ccc} x & \xrightarrow{x \leq y} & y \\ \downarrow F & & \downarrow F \\ F(x) & \xrightarrow{F(x) \leq F(y)} & F(y) \end{array}$$

- By the definition of functors,  $F$  must take each morphism  $f_{x \leq y} \in \mathbb{P}re(x, y)$  to a morphism  $F(f)_{F(x) \leq F(y)} \in \mathbb{P}re(F(x), F(y))$ .

## Exercise 1 : Order Categories, Solution c)

$$\begin{array}{c} F(x) \\ \downarrow \phi_x : F(x) \leq G(x) \\ G(x) \end{array}$$

- By the definition of natural transformations, for every object  $x$ ,  $\phi_x$  is a morphism  $F(x) \rightarrow G(x)$ . If such a morphism exists, we have  $F(x) \leq G(x)$ .

# Naturality from Polymorphism

- ▶ The naturality condition resembles an equality we saw a few weeks ago:

$$r_B \circ \text{map}(a) = \text{map}(a) \circ r_A$$

# Naturality from Polymorphism

- ▶ The naturality condition resembles an equality we saw a few weeks ago:

$$r_B \circ \text{map}(a) = \text{map}(a) \circ r_A$$

- ▶ This is the free theorem we got for a parametrically polymorphic function  $r :: [X] \rightarrow [X]$  and an arbitrary function  $a :: A \rightarrow B$ .

# Naturality from Polymorphism

- ▶ The naturality condition resembles an equality we saw a few weeks ago:

$$r_B \circ \text{map}(a) = \text{map}(a) \circ r_A$$

- ▶ This is the free theorem we got for a parametrically polymorphic function  $r :: [X] \rightarrow [X]$  and an arbitrary function  $a :: A \rightarrow B$ .
- ▶ This free theorem is equivalent to the statement that  $r$  is a natural transformation.

# Naturality from Polymorphism

- ▶ In general, assume we have:

# Naturality from Polymorphism

- ▶ In general, assume we have:
  - ▶ two functors  $F$  and  $G$ ,

# Naturality from Polymorphism

- ▶ In general, assume we have:
  - ▶ two functors  $F$  and  $G$ ,
  - ▶ a parametrically polymorphic function  $r : F\ x \rightarrow G\ x$ ,



# Naturality from Polymorphism

- ▶ In general, assume we have:
  - ▶ two functors  $F$  and  $G$ ,
  - ▶ a parametrically polymorphic function  $r : F\ x \rightarrow G\ x$ ,
  - ▶ an arbitrary function  $f : A \rightarrow B$ .

# Naturality from Polymorphism

- ▶ In general, assume we have:
  - ▶ two functors  $F$  and  $G$ ,
  - ▶ a parametrically polymorphic function  $r : F\ x \rightarrow G\ x$ ,
  - ▶ an arbitrary function  $f : A \rightarrow B$ .
- ▶ Then we get the following free theorem:

$$r \ . \ \text{fmap } f = \text{fmap } f \ . \ r$$

# Naturality from Polymorphism

- ▶ In general, assume we have:
  - ▶ two functors  $F$  and  $G$ ,
  - ▶ a parametrically polymorphic function  $r : F\ x \rightarrow G\ x$ ,
  - ▶ an arbitrary function  $f : A \rightarrow B$ .
- ▶ Then we get the following free theorem:

$$r \ . \ \text{fmap } f = \text{fmap } f \ . \ r$$

- ▶ In categorical notation:

$$r_B \circ F(f) = G(f) \circ r_A$$

# Naturality from Polymorphism

- ▶ In general, assume we have:
  - ▶ two functors  $F$  and  $G$ ,
  - ▶ a parametrically polymorphic function  $r : F\ x \rightarrow G\ x$ ,
  - ▶ an arbitrary function  $f : A \rightarrow B$ .
- ▶ Then we get the following free theorem:

$$r \ . \ \text{fmap } f = \text{fmap } f \ . \ r$$

- ▶ In categorical notation:

$$r_B \circ F(f) = G(f) \circ r_A$$

- ▶ So our free theorem is a proof that any parametrically polymorphic function  $r$  is a natural transformation!

# Naturality from Polymorphism

- ▶ In general, assume we have:
  - ▶ two functors  $F$  and  $G$ ,
  - ▶ a parametrically polymorphic function  $r : F\ x \rightarrow G\ x$ ,
  - ▶ an arbitrary function  $f : A \rightarrow B$ .
- ▶ Then we get the following free theorem:

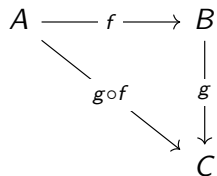
$$r \ . \ \text{fmap } f = \text{fmap } f \ . \ r$$

- ▶ In categorical notation:

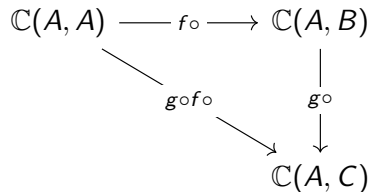
$$r_B \circ F(f) = G(f) \circ r_A$$

- ▶ So our free theorem is a proof that any parametrically polymorphic function  $r$  is a natural transformation!
- ▶ It turns out that parametrically polymorphic functions correspond exactly to natural transformations between endofunctors  $\mathbb{H}ask \rightarrow \mathbb{H}ask$ .

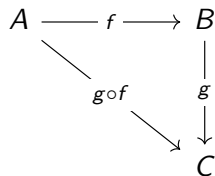
# Homfunctors



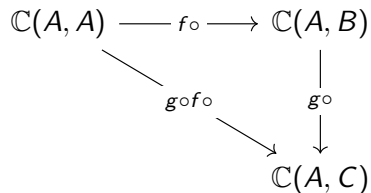
- For any locally small category  $\mathbb{C}$ , a homset  $\mathbb{C}(A, B)$  is a set of morphisms.



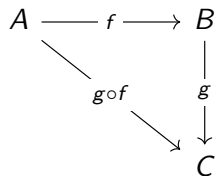
# Homfunctors



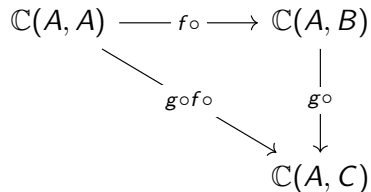
- ▶ For any locally small category  $\mathbb{C}$ , a homset  $\mathbb{C}(A, B)$  is a set of morphisms.
- ▶ We define a functor  $\mathbb{C}(A, -) : \mathbb{C} \rightarrow \mathbf{Set}$ :



# Homfunctors

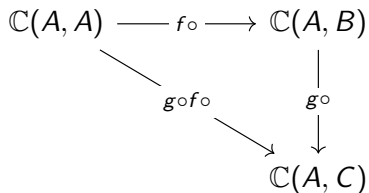
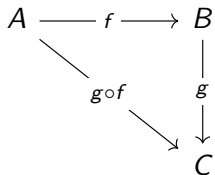


- ▶ For any locally small category  $\mathbb{C}$ , a homset  $\mathbb{C}(A, B)$  is a set of morphisms.
- ▶ We define a functor  $\mathbb{C}(A, -) : \mathbb{C} \rightarrow \mathbf{Set}$ :
  - ▶  $\mathbb{C}(A, -)$  maps an object  $B$  to the homset  $\mathbb{C}(A, B)$



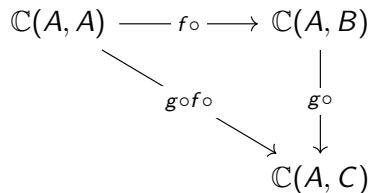
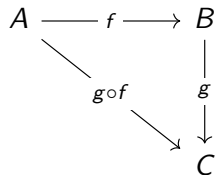


# Homfunctors



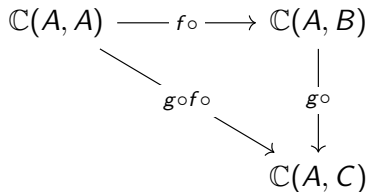
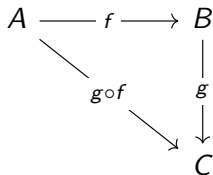
- ▶ For any locally small category  $\mathbb{C}$ , a homset  $\mathbb{C}(A, B)$  is a set of morphisms.
- ▶ We define a functor  $\mathbb{C}(A, -) : \mathbb{C} \rightarrow \mathbf{Set}$ :
  - ▶  $\mathbb{C}(A, -)$  maps an object  $B$  to the homset  $\mathbb{C}(A, B)$
  - ▶ A morphism  $f : \mathbb{C}(B, C)$  is mapped to the morphism  $f \circ : \mathbb{C}(A, B) \rightarrow \mathbb{C}(A, C)$

# Homfunctors



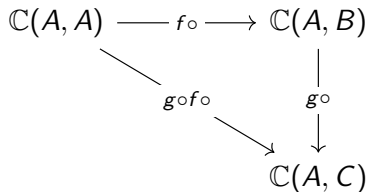
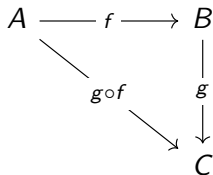
- ▶ For any locally small category  $\mathbb{C}$ , a homset  $\mathbb{C}(A, B)$  is a set of morphisms.
- ▶ We define a functor  $\mathbb{C}(A, -) : \mathbb{C} \rightarrow \mathbf{Set}$ :
  - ▶  $\mathbb{C}(A, -)$  maps an object  $B$  to the homset  $\mathbb{C}(A, B)$
  - ▶ A morphism  $f : \mathbb{C}(B, C)$  is mapped to the morphism  $f \circ : \mathbb{C}(A, B) \rightarrow \mathbb{C}(A, C)$
- ▶ Similarly,  $\mathbb{C}(-, B)$  is a functor  $\mathbb{C}^{op} \rightarrow \mathbf{Set}$ :

# Homfunctors



- ▶ For any locally small category  $\mathbb{C}$ , a homset  $\mathbb{C}(A, B)$  is a set of morphisms.
- ▶ We define a functor  $\mathbb{C}(A, -) : \mathbb{C} \rightarrow \mathbf{Set}$ :
  - ▶  $\mathbb{C}(A, -)$  maps an object  $B$  to the homset  $\mathbb{C}(A, B)$
  - ▶ A morphism  $f : \mathbb{C}(B, C)$  is mapped to the morphism  $f \circ : \mathbb{C}(A, B) \rightarrow \mathbb{C}(A, C)$
- ▶ Similarly,  $\mathbb{C}(-, B)$  is a functor  $\mathbb{C}^{op} \rightarrow \mathbf{Set}$ :
  - ▶  $\mathbb{C}(-, B)$  maps an object  $A$  to the homset  $\mathbb{C}(A, B)$

# Homfunctors



- ▶ For any locally small category  $\mathbb{C}$ , a homset  $\mathbb{C}(A, B)$  is a set of morphisms.
- ▶ We define a functor  $\mathbb{C}(A, -) : \mathbb{C} \rightarrow \mathbf{Set}$ :
  - ▶  $\mathbb{C}(A, -)$  maps an object  $B$  to the homset  $\mathbb{C}(A, B)$
  - ▶ A morphism  $f : \mathbb{C}(B, C)$  is mapped to the morphism  $f \circ : \mathbb{C}(A, B) \rightarrow \mathbb{C}(A, C)$
- ▶ Similarly,  $\mathbb{C}(-, B)$  is a functor  $\mathbb{C}^{op} \rightarrow \mathbf{Set}$ :
  - ▶  $\mathbb{C}(-, B)$  maps an object  $A$  to the homset  $\mathbb{C}(A, B)$
  - ▶ A morphism  $f : \mathbb{C}(C, A)$  is mapped to the morphism  $\circ f : \mathbb{C}(A, B) \rightarrow \mathbb{C}(C, B)$

# Functor Categories

- ▶ For any  $\mathbb{C}$ ,  $\mathbb{D}$ , the collection of functors  $\mathbb{C} \rightarrow \mathbb{D}$  form a category.

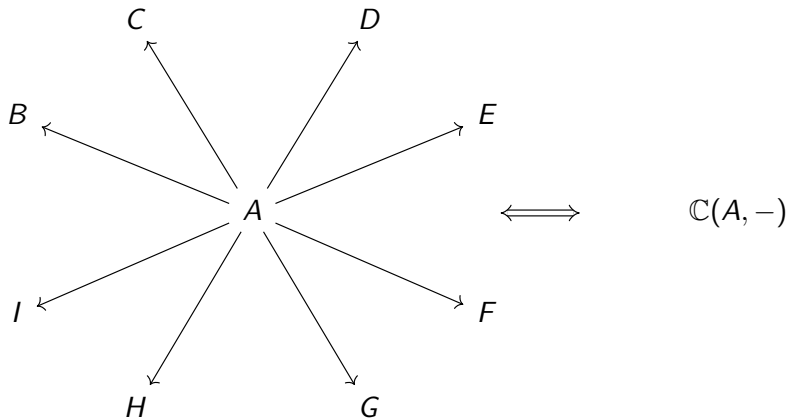
# Functor Categories

- ▶ For any  $\mathbb{C}$ ,  $\mathbb{D}$ , the collection of functors  $\mathbb{C} \rightarrow \mathbb{D}$  form a category.
- ▶ This category is known as a *functor category* and denoted  $\mathbb{D}^{\mathbb{C}}$ .

# Functor Categories

- ▶ For any  $\mathbb{C}$ ,  $\mathbb{D}$ , the collection of functors  $\mathbb{C} \rightarrow \mathbb{D}$  form a category.
- ▶ This category is known as a *functor category* and denoted  $\mathbb{D}^{\mathbb{C}}$ .
- ▶ A morphism  $\phi \in \mathbb{D}^{\mathbb{C}}(F, G)$  is a natural transformation  $F \rightarrow G$ .

# The Yoneda Embedding





# The Yoneda Embedding

- Formally, we want a bijective functor

$$\mathcal{Y} : \mathbb{C} \rightarrow \mathbf{Set}^{\mathbb{C}}$$

$$A \mapsto \mathbb{C}(A, -)$$

$$\begin{array}{ccc} A & \xrightarrow{f \in \mathbb{C}(A, B)} & B \\ \uparrow \scriptstyle \mathcal{Y} & & \uparrow \scriptstyle \mathcal{Y} \\ \mathbb{C}(A, -) & \xrightarrow{\mathcal{Y}(f)} & \mathbb{C}(B, -) \end{array}$$

# The Yoneda Embedding

- Formally, we want a bijective functor

$$\mathcal{Y} : \mathbb{C} \rightarrow \mathbf{Set}^{\mathbb{C}}$$
$$A \mapsto \mathbb{C}(A, -)$$

$$\begin{array}{ccc} A & \xrightarrow{f \in \mathbb{C}(A, B)} & B \\ \mathcal{Y} \uparrow \text{dotted} & & \mathcal{Y} \uparrow \text{dotted} \\ \mathbb{C}(A, -) & \xrightarrow{\mathcal{Y}(f)} & \mathbb{C}(B, -) \end{array}$$

- We call  $\mathcal{Y}$  the *Yoneda embedding*.

# The Yoneda Embedding

- Formally, we want a bijective functor

$$\mathcal{Y} : \mathbb{C} \rightarrow \mathbf{Set}^{\mathbb{C}}$$
$$A \mapsto \mathbb{C}(A, -)$$

$$\begin{array}{ccc} A & \xrightarrow{f \in \mathbb{C}(A, B)} & B \\ \mathcal{Y} \uparrow & & \uparrow \mathcal{Y} \\ \mathbb{C}(A, -) & \xrightarrow{\mathcal{Y}(f)} & \mathbb{C}(B, -) \end{array}$$

- We call  $\mathcal{Y}$  the *Yoneda embedding*.
- Given  $f \in \mathbb{C}(A, B)$ ,  $\mathcal{Y}(f)$  has to be a morphism between  $\mathbb{C}(A, -)$  and  $\mathbb{C}(B, -)$  in the functor category  $\mathbf{Set}^{\mathbb{C}}$ .

# The Yoneda Embedding

- Formally, we want a bijective functor

$$\mathcal{Y} : \mathbb{C} \rightarrow \mathbf{Set}^{\mathbb{C}}$$
$$A \mapsto \mathbb{C}(A, -)$$

$$\begin{array}{ccc} A & \xrightarrow{f \in \mathbb{C}(A, B)} & B \\ \mathcal{Y} \uparrow \downarrow & & \uparrow \downarrow \mathcal{Y} \\ \mathbb{C}(A, -) & \xrightarrow{\mathcal{Y}(f)} & \mathbb{C}(B, -) \end{array}$$

- We call  $\mathcal{Y}$  the *Yoneda embedding*.
- Given  $f \in \mathbb{C}(A, B)$ ,  $\mathcal{Y}(f)$  has to be a morphism between  $\mathbb{C}(A, -)$  and  $\mathbb{C}(B, -)$  in the functor category  $\mathbf{Set}^{\mathbb{C}}$ .
- Therefore,  $\mathcal{Y}(f)$  has to be a natural transformation between  $\mathbb{C}(A, -)$  and  $\mathbb{C}(B, -)$ .

# The Yoneda Embedding

- Formally, we want a bijective functor

$$\mathcal{Y} : \mathbb{C} \rightarrow \mathbf{Set}^{\mathbb{C}}$$
$$A \mapsto \mathbb{C}(A, -)$$

$$\begin{array}{ccc} A & \xrightarrow{f \in \mathbb{C}(A, B)} & B \\ \mathcal{Y} \uparrow \text{dotted} & & \uparrow \text{dotted} \mathcal{Y} \\ \mathbb{C}(A, -) & \xrightarrow{\mathcal{Y}(f)} & \mathbb{C}(B, -) \end{array}$$

- We call  $\mathcal{Y}$  the *Yoneda embedding*.
- Given  $f \in \mathbb{C}(A, B)$ ,  $\mathcal{Y}(f)$  has to be a morphism between  $\mathbb{C}(A, -)$  and  $\mathbb{C}(B, -)$  in the functor category  $\mathbf{Set}^{\mathbb{C}}$ .
- Therefore,  $\mathcal{Y}(f)$  has to be a natural transformation between  $\mathbb{C}(A, -)$  and  $\mathbb{C}(B, -)$ .
- Is it actually possible to construct all of the necessary natural transformations?

# The Yoneda lemma

- ▶ It turns out we can do even better!

# The Yoneda lemma

- ▶ It turns out we can do even better!
- ▶ We can construct the set of all natural transformations between  $\mathbb{C}(A, -)$  and any Functor  $F : \mathbb{C} \rightarrow \mathbf{Set}$ .

# The Yoneda lemma

- ▶ It turns out we can do even better!
- ▶ We can construct the set of all natural transformations between  $\mathbb{C}(A, -)$  and any Functor  $F : \mathbb{C} \rightarrow \mathbf{Set}$ .
- ▶ Specifically, the Yoneda lemma states that:

$$\mathrm{Nat}(\mathbb{C}(A, -), F) \simeq F(A)$$



# The Yoneda lemma

- ▶ It turns out we can do even better!
- ▶ We can construct the set of all natural transformations between  $\mathbb{C}(A, -)$  and any Functor  $F : \mathbb{C} \rightarrow \mathbf{Set}$ .
- ▶ Specifically, the Yoneda lemma states that:

$$\mathrm{Nat}(\mathbb{C}(A, -), F) \simeq F(A)$$

- ▶ Furthermore, this isomorphism is a natural transformation.

# The Yoneda lemma

- ▶ It turns out we can do even better!
- ▶ We can construct the set of all natural transformations between  $\mathbb{C}(A, -)$  and any Functor  $F : \mathbb{C} \rightarrow \mathbb{Set}$ .
- ▶ Specifically, the Yoneda lemma states that:

$$\text{Nat}(\mathbb{C}(A, -), F) \simeq F(A)$$

- ▶ Furthermore, this isomorphism is a natural transformation.
- ▶ So we can construct the Yoneda embedding  $\mathcal{Y}$  from the set  $F(A)$ .

# The Yoneda lemma

- ▶ It turns out we can do even better!
- ▶ We can construct the set of all natural transformations between  $\mathbb{C}(A, -)$  and any Functor  $F : \mathbb{C} \rightarrow \mathbb{Set}$ .
- ▶ Specifically, the Yoneda lemma states that:

$$\text{Nat}(\mathbb{C}(A, -), F) \simeq F(A)$$

- ▶ Furthermore, this isomorphism is a natural transformation.
- ▶ So we can construct the Yoneda embedding  $\mathcal{Y}$  from the set  $F(A)$ .
- ▶ Vice versa, if we know all natural transformations  $\text{Nat}(\mathbb{C}(A, -), F)$ , we can construct the set  $F(A)$ .

## Constructing the bijection

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\[1em] \mathbb{C}(A, A) & \xrightarrow{\mathbb{C}(A, f) = f \circ} & \mathbb{C}(A, B) \\ \downarrow \phi_A & & \downarrow \phi_B \\ F(A) & \xrightarrow{F(f)} & F(B) \end{array}$$

# Constructing the bijection

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ \mathbb{C}(A, A) & \xrightarrow{\mathbb{C}(A, f) = f \circ} & \mathbb{C}(A, B) \\ \downarrow \phi_A & & \downarrow \phi_B \\ F(A) & \xrightarrow{F(f)} & F(B) \end{array}$$

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ id_A & \xrightarrow{\mathbb{C}(A, f) = f \circ} & f \\ \downarrow \phi_A & & \downarrow \phi_B \\ u & \xrightarrow{F(f)} & \phi_B(f) = F(f)(u) \end{array}$$

## Application : Cayley's Theorem

- ▶ Every group  $(G, *, e)$  is isomorphic to a subgroup of the group of permutations of  $G$ .

## Application : Cayley's Theorem

- ▶ Every group  $(G, *, e)$  is isomorphic to a subgroup of the group of permutations of  $G$ .
- ▶ Specifically:
  - ▶ One side of the bijection is constructed by sending  $g \in G$  to the permutation which maps  $f_g : x \mapsto g * x$
  - ▶ The other side sends a permutation  $f$  to the element  $f(e)$

## Application : Cayley's Theorem

- ▶ Every group  $(G, *, e)$  is isomorphic to a subgroup of the group of permutations of  $G$ .
- ▶ Specifically:
  - ▶ One side of the bijection is constructed by sending  $g \in G$  to the permutation which maps  $f_g : x \mapsto g * x$
  - ▶ The other side sends a permutation  $f$  to the element  $f(e)$
- ▶ The Yoneda lemma is often viewed as a generalization of Cayley's theorem.



## Exercise 2 - Cayley's Theorem for Monoids

Use the Yoneda embedding to show that every monoid  $M$  is isomorphic to a monoid of functions  $M \rightarrow M$ .

**Hint 1:** The Yoneda embedding gives an isomorphism between objects and their homfunctors.

**Hint 2:** Two weeks ago we saw that every monoid  $M$  defines a category  $\mathbb{M}$  with a single object  $*$  and a morphism  $m$  for each element  $m \in M$ , where we define morphism composition to be the monoid operation.

## Exercise 2 - Cayley's Theorem for Monoids

- The Yoneda embedding is an isomorphism mapping each object to its homfunctor.

$$\begin{array}{ccc} * & \xrightarrow{m \in \mathbb{M}(*,*)} & * \\ \uparrow \mathcal{Y} & & \uparrow \mathcal{Y} \\ \mathbb{M}(*, -) & \xrightarrow{\mathcal{Y}(m) = m \circ} & \mathbb{M}(*, -) \end{array}$$

## Exercise 2 - Cayley's Theorem for Monoids

- ▶ The Yoneda embedding is an isomorphism mapping each object to its homfunctor.
- ▶ We only have one object  $*$ , and thus only one homfunctor  $\mathbb{M}(*, -)$ .

$$\begin{array}{ccc} * & \xrightarrow{m \in \mathbb{M}(*,*)} & * \\ \uparrow \mathcal{Y} & & \uparrow \mathcal{Y} \\ \mathbb{M}(*, -) & \xrightarrow{\mathcal{Y}(m)=m \circ} & \mathbb{M}(*, -) \end{array}$$

## Exercise 2 - Cayley's Theorem for Monoids

- ▶ The Yoneda embedding is an isomorphism mapping each object to its homfunctor.
- ▶ We only have one object  $*$ , and thus only one homfunctor  $\mathbb{M}(*, -)$ .
- ▶ Each element  $m \in M$  is a morphism. By the definition of the homfunctor, this morphism is mapped to the set function

$$\begin{array}{ccc} * & \xrightarrow{m \in \mathbb{M}(*,*)} & * \\ \uparrow & & \uparrow \\ \mathcal{Y} & & \mathcal{Y} \\ \downarrow & & \downarrow \\ \mathbb{M}(*, -) & \xrightarrow{\mathcal{Y}(m)=m \circ} & \mathbb{M}(*, -) \end{array}$$

$$\begin{aligned} \mathbb{M}(*, m) &= m \circ : M \rightarrow M \\ n &\mapsto m \circ n \end{aligned}$$

## Exercise 2 - Cayley's Theorem for Monoids

$$\begin{array}{ccc} * & \xrightarrow{m \in \mathbb{M}(*,*)} & * \\ \uparrow & & \uparrow \\ \mathcal{Y} & & \mathcal{Y} \\ \downarrow & & \downarrow \\ \mathbb{M}(*, -) & \xrightarrow{\mathcal{Y}(m) = m \circ} & \mathbb{M}(*, -) \end{array}$$

- ▶ The Yoneda embedding is an isomorphism mapping each object to its homfunctor.
- ▶ We only have one object  $*$ , and thus only one homfunctor  $\mathbb{M}(*, -)$ .
- ▶ Each element  $m \in M$  is a morphism. By the definition of the homfunctor, this morphism is mapped to the set function

$$\begin{aligned} \mathbb{M}(*, m) &= m \circ : M \rightarrow M \\ n &\mapsto m \circ n \end{aligned}$$

- ▶ Thus, the Yoneda embedding on  $M$  is an isomorphism between monoid objects and a set of functions  $M \rightarrow M$ . These functions form a monoid under composition.



# Profunctor Optics

- ▶ In functional programming, an *optic* is generally a data structure including some "outer type"  $S$  and some "inner type"  $A$ .

# Profunctor Optics

- ▶ In functional programming, an *optic* is generally a data structure including some "outer type"  $S$  and some "inner type"  $A$ .
- ▶ In general, this involves some sort of get function, which allows access to the inner value, and a set function, which changes the inner value.

# Profunctor Optics

- ▶ In functional programming, an *optic* is generally a data structure including some "outer type"  $S$  and some "inner type"  $A$ .
- ▶ In general, this involves some sort of get function, which allows access to the inner value, and a set function, which changes the inner value.
- ▶ *Profunctor optics* are neat and flexible representations of optics as individual polymorphic function.



# Profunctor Optics

- ▶ In functional programming, an *optic* is generally a data structure including some "outer type"  $S$  and some "inner type"  $A$ .
- ▶ In general, this involves some sort of get function, which allows access to the inner value, and a set function, which changes the inner value.
- ▶ *Profunctor optics* are neat and flexible representations of optics as individual polymorphic function.
- ▶ In particular, profunctor optics make composition of optics trivial.

# Profunctor Optics

- ▶ In functional programming, an *optic* is generally a data structure including some "outer type"  $S$  and some "inner type"  $A$ .
- ▶ In general, this involves some sort of get function, which allows access to the inner value, and a set function, which changes the inner value.
- ▶ *Profunctor optics* are neat and flexible representations of optics as individual polymorphic function.
- ▶ In particular, profunctor optics make composition of optics trivial.
- ▶ Equivalence between optics and their profunctor representations comes down to the Yoneda lemma.

# Adapters

```
data Adapter a b s t = Adapter { from :: s -> a, to :: b -> t }
```

# Adapters

`data Adapter a b s t = Adapter { from :: s -> a, to :: b -> t }`

- Categorically, this translates to a pair of morphisms  $\mathbb{S}et(S, A) \times \mathbb{S}et(B, T)$ .

# Adapters

`data Adapter a b s t = Adapter { from :: s -> a, to :: b -> t }`

- ▶ Categorically, this translates to a pair of morphisms  $\mathbb{S}et(S, A) \times \mathbb{S}et(B, T)$ .
- ▶ This pair is a morphism  $(\mathbb{S}et^{op} \times \mathbb{S}et)((A, B), (S, T))$ .

# Adapters

```
data Adapter a b s t = Adapter { from :: s -> a, to :: b -> t }
```

- ▶ Categorically, this translates to a pair of morphisms  $\mathbb{S}et(S, A) \times \mathbb{S}et(B, T)$ .
- ▶ This pair is a morphism  $(\mathbb{S}et^{op} \times \mathbb{S}et)((A, B), (S, T))$ .
- ▶ We can compose adapters with matching types and define an identity adapter `Adapter id id`.

# Adapters

```
data Adapter a b s t = Adapter { from :: s -> a, to :: b -> t }
```

- ▶ Categorically, this translates to a pair of morphisms  $\text{Set}(S, A) \times \text{Set}(B, T)$ .
- ▶ This pair is a morphism  $(\text{Set}^{op} \times \text{Set})((A, B), (S, T))$ .
- ▶ We can compose adapters with matching types and define an identity adapter `Adapter id id`.
- ▶ This lets us view the category  $\text{Set}^{op} \times \text{Set}$  as the category  $\mathbb{A}da$  where morphisms are adapters.

# Profunctors

- ▶ A *profunctor* is a functor  $\mathbb{C}^{op} \times \mathbb{C} \rightarrow \mathbf{Set}$ .



# Profunctors

► A *profunctor* is a functor  $\mathbb{C}^{op} \times \mathbb{C} \rightarrow \mathbf{Set}$ .

► As Haskell code:

```
class Profunctor p where  
  dimap :: (s -> a) -> (b -> t) -> p a b -> p s t
```

# Profunctors

► A *profunctor* is a functor  $\mathbb{C}^{op} \times \mathbb{C} \rightarrow \mathbf{Set}$ .

► As Haskell code:

```
class Profunctor p where  
  dimap :: (s -> a) -> (b -> t) -> p a b -> p s t
```

► The canonical example of a profunctor is the function type former, where `dimap` is function composition:

```
instance Profunctor (->) where  
  dimap f g h = g . h . f
```

# Profunctors

- ▶ A *profunctor* is a functor  $\mathbb{C}^{op} \times \mathbb{C} \rightarrow \mathbf{Set}$ .

- ▶ As Haskell code:

```
class Profunctor p where  
  dimap :: (s -> a) -> (b -> t) -> p a b -> p s t
```

- ▶ The canonical example of a profunctor is the function type former, where `dimap` is function composition:

```
instance Profunctor (->) where  
  dimap f g h = g . h . f
```

- ▶ We define the category  $\mathbb{P}rof$  of Profunctors to be  $\mathbf{Set}^{\mathbf{Set}^{op} \times \mathbf{Set}} = \mathbf{Set}^{Ada}$

# Functor Application as a Functor

$$\begin{array}{ccc} F & \xrightarrow{\eta} & G \\ \text{\tiny $-(A)$} \downarrow \text{\tiny $\vee$} & & \downarrow \text{\tiny $-(A)$} \text{\tiny $\vee$} \\ F(A) & \xrightarrow{\eta_A} & G(A) \end{array}$$

- ▶ Given a category  $\mathbb{C}$ , the operation of applying a functor  $F : \mathbb{C} \rightarrow \mathbf{Set}$  to an object  $A \in |\mathbb{C}|$  is itself a functor from  $\mathbf{Set}^{\mathbb{C}}$  to  $\mathbf{Set}$ .

# Functor Application as a Functor

$$\begin{array}{ccc} F & \xrightarrow{\eta} & G \\ \text{\tiny $-(A)$} \downarrow \text{\tiny $\vee$} & & \downarrow \text{\tiny $-(A)$} \text{\tiny $\vee$} \\ F(A) & \xrightarrow{\eta_A} & G(A) \end{array}$$

- ▶ Given a category  $\mathbb{C}$ , the operation of applying a functor  $F : \mathbb{C} \rightarrow \mathbf{Set}$  to an object  $A \in |\mathbb{C}|$  is itself a functor from  $\mathbf{Set}^{\mathbb{C}}$  to  $\mathbf{Set}$ .
- ▶ We write  $-(A)$  for this functor.

# Functor Application as a Functor

$$\begin{array}{ccc} F & \xrightarrow{\eta} & G \\ \text{\tiny $-(A)$} \downarrow \text{\tiny $\vee$} & & \downarrow \text{\tiny $-(A)$} \text{\tiny $\vee$} \\ F(A) & \xrightarrow{\eta_A} & G(A) \end{array}$$

- ▶ Given a category  $\mathbb{C}$ , the operation of applying a functor  $F : \mathbb{C} \rightarrow \mathbf{Set}$  to an object  $A \in |\mathbb{C}|$  is itself a functor from  $\mathbf{Set}^{\mathbb{C}}$  to  $\mathbf{Set}$ .
- ▶ We write  $-(A)$  for this functor.
- ▶  $-(A) \in \mathbf{Set}^{\mathbf{Set}^{\mathbb{C}}}$

# Profunctor Adapters

- The profunctor representation of an adapter is given by:

`type AdapterP a b s t =`  
    forall p. Profunctor p => p a b => p s t

$$\begin{array}{ccc} p(A, B) & \xrightarrow{\eta} & p'(A, B) \\ \text{AdapterP}_p \downarrow & & \downarrow \text{AdapterP}_{p'} \\ p(S, T) & \xrightarrow{\vartheta} & p'(S, T) \end{array}$$

# Profunctor Adapters

- ▶ The profunctor representation of an adapter is given by:

```
type AdapterP a b s t =  
  forall p. Profunctor p => p a b -> p s t
```

- ▶ As we saw earlier, the polymorphism makes AdapterP a natural transformation.

$$\begin{array}{ccc} p(A, B) - \eta \rightarrow p'(A, B) & & \\ \text{AdapterP}_p \downarrow & & \text{AdapterP}_{p'} \downarrow \\ p(S, T) - \vartheta \rightarrow p'(S, T) & & \end{array}$$



# Profunctor Adapters

- ▶ The profunctor representation of an adapter is given by:

```
type AdapterP a b s t =  
  forall p. Profunctor p => p a b -> p s t
```

- ▶ As we saw earlier, the polymorphism makes `AdapterP` a natural transformation.
- ▶ We define  $\mathbb{A}daP$  as the functor category where objects are  $|\mathbb{A}daP| = |\mathbb{A}da| = |\mathbf{Set}^{op} \times \mathbf{Set}|$ , and whose morphisms are profunctor adapters.

$$\begin{array}{ccc} p(A, B) - \eta \rightarrow p'(A, B) & & \\ \text{AdapterP}_p \downarrow & \text{AdapterP}_{p'} \downarrow & \\ p(S, T) - \vartheta \rightarrow p'(S, T) & & \end{array}$$

# Profunctor Adapters

- ▶ The profunctor representation of an adapter is given by:

```
type AdapterP a b s t =  
  forall p. Profunctor p => p a b -> p s t
```

- ▶ As we saw earlier, the polymorphism makes `AdapterP` a natural transformation.

- ▶ We define  $\mathbb{A}daP$  as the functor category where objects are  $|\mathbb{A}daP| = |\mathbb{A}da| = |\mathbf{Set}^{op} \times \mathbf{Set}|$ , and whose morphisms are profunctor adapters.

- ▶ Specifically, the homsets in  $\mathbb{A}daP$  are:

$$\begin{aligned}\mathbb{A}daP((A, B), (S, T)) &= \mathbf{Set}^{\mathbf{Set}^{\mathbb{A}da}}(-(A, B), -(S, T)) \\ &= \mathbf{Nat}(-(A, B), -(S, T))\end{aligned}$$

$$\begin{array}{ccc} p(A, B) - \eta \rightarrow p'(A, B) & & \\ \text{AdapterP}_p \downarrow & & \text{AdapterP}_{p'} \downarrow \\ p(S, T) - \vartheta \rightarrow p'(S, T) & & \end{array}$$

## Equivalence of the representations

$$\mathbb{A}daP((A, B), (S, T)) = \text{Nat}(-(A, B), -(S, T)) \stackrel{?}{\simeq} \mathbb{A}da((A, B), (S, T))$$

## Equivalence of the representations

$$\mathbb{A}daP((A, B), (S, T)) = \text{Nat}(-(A, B), -(S, T)) \stackrel{?}{\simeq} \mathbb{A}da((A, B), (S, T))$$

- This involves expanding the functors on the left using the Yoneda lemma and then summarizing everything by applying the Yoneda embedding twice.

## Equivalence of the representations

$$\mathbb{A}daP((A, B), (S, T)) = \text{Nat}(-(A, B), -(S, T)) \stackrel{?}{\simeq} \mathbb{A}da((A, B), (S, T))$$

- ▶ This involves expanding the functors on the left using the Yoneda lemma and then summarizing everything by applying the Yoneda embedding twice.
- ▶ This proof is left as an exercise to the reader :)

## Equivalence of the representations

$$\mathbb{A}daP((A, B), (S, T)) = \text{Nat}(-(A, B), -(S, T)) \stackrel{?}{\simeq} \mathbb{A}da((A, B), (S, T))$$

- ▶ This involves expanding the functors on the left using the Yoneda lemma and then summarizing everything by applying the Yoneda embedding twice.
- ▶ This proof is left as an exercise to the reader :) (a short version of the proof is uploaded alongside the slides on ilias).

# Summary

- ▶ The Yoneda lemma is a fundamental result in category theory.

# Summary

- ▶ The Yoneda lemma is a fundamental result in category theory.
- ▶ The lemma states that the set of natural transformations between a homfunctor  $\mathbb{C}(A, -)$  and an arbitrary functor  $F : \mathbb{C} \rightarrow \mathbf{Set}$  is naturally isomorphic to the set  $F(A)$ .



# Summary

- ▶ The Yoneda lemma is a fundamental result in category theory.
- ▶ The lemma states that the set of natural transformations between a homfunctor  $\mathbb{C}(A, -)$  and an arbitrary functor  $F : \mathbb{C} \rightarrow \mathbf{Set}$  is naturally isomorphic to the set  $F(A)$ .
- ▶ The Yoneda lemma enables the use of the Yoneda embedding, which is a natural isomorphism between a category  $\mathbb{C}$  and the category of homfunctors on  $\mathbb{C}$ .

# Summary

- ▶ The Yoneda lemma is a fundamental result in category theory.
- ▶ The lemma states that the set of natural transformations between a homfunctor  $\mathbb{C}(A, -)$  and an arbitrary functor  $F : \mathbb{C} \rightarrow \mathbf{Set}$  is naturally isomorphic to the set  $F(A)$ .
- ▶ The Yoneda lemma enables the use of the Yoneda embedding, which is a natural isomorphism between a category  $\mathbb{C}$  and the category of homfunctors on  $\mathbb{C}$ .
- ▶ Equivalence of adapters and profunctor adapters can be shown by applying the Yoneda embedding twice.

# Summary

- ▶ The Yoneda lemma is a fundamental result in category theory.
- ▶ The lemma states that the set of natural transformations between a homfunctor  $\mathbb{C}(A, -)$  and an arbitrary functor  $F : \mathbb{C} \rightarrow \mathbf{Set}$  is naturally isomorphic to the set  $F(A)$ .
- ▶ The Yoneda lemma enables the use of the Yoneda embedding, which is a natural isomorphism between a category  $\mathbb{C}$  and the category of homfunctors on  $\mathbb{C}$ .
- ▶ Equivalence of adapters and profunctor adapters can be shown by applying the Yoneda embedding twice.
- ▶ Similar techniques can be used to show the equivalence of any optic and its profunctor representation.

Thank You!