universität freiburg

$$
\begin{tikzcd}
\mathrm{Hom}(A,A) \arrow[r, "\mathrm{Hom}(A,f)"] \arrow[d, "\Phi_A"'] & \mathrm{Hom}(A,X) \arrow[d, "\Phi_X"] \\
F(A) \arrow[r, "Ff"'] & F(X)
\end{tikzcd}
$$

$$
\begin{tikzcd}
\mathrm{id}_A \arrow[r, mapsto] \arrow[d, mapsto] & f \arrow[d, mapsto] \\
u \arrow[r, mapsto] & (Ff)u = \Phi_X(f)
\end{tikzcd}
$$

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.

[1] 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.*"[1]

[1] 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.*"[1]
- The Yoneda Lemma is the result of applying this way of thinking to mathematical objects in category theory.

[1] 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.*"[1]
- 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 $\mathbb{S}et$.

[1] Quoted as a proverb in *Don Quixote*

Categories

$$
\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "g\circ f"'] \arrow[loop above, "id_1"] & B \arrow[d, "g"] \arrow[loop above, "id_2"] \\
 & C \arrow[loop below, "id_3"]
\end{tikzcd}
$$

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

Categories

$$
\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "g \circ f"'] \arrow[loop above, "id_1"] & B \arrow[d, "g"] \arrow[loop above, "id_2"] \\
& C \arrow[loop below, "id_3"]
\end{tikzcd}
$$

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

Categories

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[loop above, "id_1"] \arrow[dr, "g \circ f"'] & B \arrow[d, "g"] \arrow[loop above, "id_2"] \\
& C \arrow[loop below, "id_3"]
\end{tikzcd}$$

- 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;

Categories

$$\begin{tikzcd}
A \arrow[loop above, "id_1"] \arrow[r, "f"] \arrow[dr, "g \circ f"'] & B \arrow[loop above, "id_2"] \arrow[d, "g"] \\
& C \arrow[loop below, "id_3"]
\end{tikzcd}$$

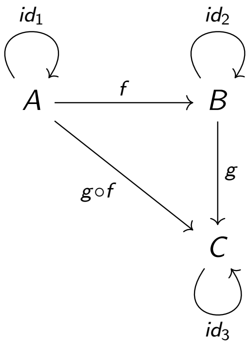
- A *category* $\mathbb{C}$ consists of:
  - a collection $|\mathbb{C}|$ of *objects*;
  - for all $A, B \in |\mathbb{C}|$, a collection $\mathbb{C}(A, B)$ of *morphisms* from A to B;
  - for all $A \in |\mathbb{C}|$, an *identity morphism* $id_A \in \mathbb{C}(A, A)$;

Categories

$$
\begin{tikzcd}
A \arrow[r, "f"] \arrow[loop above, "id_1"] \arrow[dr, "g\circ f"'] & B \arrow[d, "g"] \arrow[loop above, "id_2"] \\
& C \arrow[loop below, "id_3"]
\end{tikzcd}
$$

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

Categories

$$\begin{tikzcd}
A \arrow[loop above, "id_1"] \arrow[r, "f"] \arrow[dr, "g \circ f"'] & B \arrow[loop above, "id_2"] \arrow[d, "g"] \\
 & C \arrow[loop below, "id_3"]
\end{tikzcd}$$

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

Categories

$$
\begin{tikzcd}
A \arrow[r, "f"] \arrow[loop above, "id_1"] \arrow[dr, "g \circ f"'] & B \arrow[d, "g"] \arrow[loop above, "id_2"] \\
& C \arrow[loop below, "id_3"]
\end{tikzcd}
$$

- 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;
  - for all $A \in |\mathbb{C}|$, an *identity morphism* $id_A \in \mathbb{C}(A, A)$;
  - for each pair of morphisms $g \in \mathbb{C}(A, B)$, $f \in \mathbb{C}(A, B)$, a morphism $g \circ f \in \mathbb{C}(A, C)$, such that composition is associative.
- If $\mathbb{C}(A, B)$ is a set, we call it the *homset* from A to B.
- 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* $F : \mathbb{C} \to \mathbb{D}$ is a structure-preserving map between two categories:

$$
\begin{tikzcd}
A \arrow[r, "f \,\in\, \mathbb{C}(A{,}B)"] \arrow[d, dotted, "F"'] & B \arrow[d, dotted, "F"] \\
F(A) \arrow[r, "F(f)"'] & F(A)
\end{tikzcd}
$$

Functors

$$\begin{tikzcd} A \arrow[r, "f \,\in\, \mathbb{C}(A{,}B)"] \arrow[d, dotted, "F"'] & B \arrow[d, dotted, "F"] \\ F(A) \arrow[r, "F(f)"'] & F(A) \end{tikzcd}$$

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

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

Functors

$$\begin{tikzcd}
A \arrow[r, "f \,\in\, \mathbb{C}(A{,}B)"] \arrow[d, dotted, "F"'] & B \arrow[d, dotted, "F"] \\
F(A) \arrow[r, "F(f)"'] & F(A)
\end{tikzcd}$$

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

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

Functors

$$
\begin{tikzcd}
A \arrow[r, "f \in \mathbb{C}(A{,}B)"] \arrow[d, dotted, "F"'] & B \arrow[d, dotted, "F"] \\
F(A) \arrow[r, "F(f)"'] & F(A)
\end{tikzcd}
$$

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

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

$$\begin{tikzcd}[column sep=huge, row sep=huge]
A \arrow[r, "f \in \mathbb{C}(A{,}B)"] \arrow[d, dotted, "F"'] & B \arrow[d, dotted, "F"] \\
F(A) \arrow[r, "F(f)"'] & F(A)
\end{tikzcd}$$

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

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

$$A \xrightarrow{f} B$$

$$\begin{tikzcd} F(A) \arrow[r, "F(f)"] \arrow[d, "\phi_A"] & F(B) \arrow[d, "\phi_B"] \\ G(A) \arrow[r, "G(f)"] & G(B) \end{tikzcd}$$

- Structure-preserving maps between functors.

Natural Transformations

$$A \xrightarrow{f} B$$

$$\begin{tikzcd}
F(A) \arrow[r, "F(f)"] \arrow[d, "\phi_A"] & F(B) \arrow[d, "\phi_B"] \\
G(A) \arrow[r, "G(f)"] & G(B)
\end{tikzcd}$$

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

Natural Transformations

$$A \xrightarrow{f} B$$

$$\begin{tikzcd}
F(A) \arrow[r, "F(f)"] \arrow[d, "\phi_A"] & F(B) \arrow[d, "\phi_B"] \\
G(A) \arrow[r, "G(f)"] & G(B)
\end{tikzcd}$$

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

Natural Transformations

$$\begin{tikzcd} A \arrow[r, "f"] & B \end{tikzcd}$$

$$\begin{tikzcd} F(A) \arrow[r, "F(f)"] \arrow[d, "\phi_A"] & F(B) \arrow[d, "\phi_B"] \\ G(A) \arrow[r, "G(f)"] & G(B) \end{tikzcd}$$

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

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

Natural Transformations

$$A \xrightarrow{f} B$$

$$\begin{tikzcd} F(A) \arrow[r, "F(f)"] \arrow[d, "\phi_A"] & F(B) \arrow[d, "\phi_B"] \\ G(A) \arrow[r, "G(f)"] & G(B) \end{tikzcd}$$

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

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

- Given two functors F and G, we write the collection of all natural transformation between them as $\mathrm{Nat}(F, G)$.

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 $F : \mathbb{M} \to \mathbb{M}$ be an endofunctor on $\mathbb{M}$. Show that F defines a monotonic function $M \to M$, i.e. $\forall x, y : x \leq y \implies F(x) \leq F(y)$.

c) Let $F, G : M \to M$ be monotonic functions. Let ϕ be a natural transformation $F \to G$. Show that $\forall x \in M : F(x) \leq G(x)$.

Exercise 1 : Order Categories, Solution a)

$$
\begin{tikzcd}
x \arrow[loop above, "x\leq x"] \arrow[r, "x\leq y"'] \arrow[dr, "x\leq z"'] & y \arrow[loop above, "y\leq y"] \arrow[d, "y\leq z"'] \\
& z \arrow[loop below, "z\leq z"]
\end{tikzcd}
$$

- $\leq$ 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 dont care about the order of function application, so composition is associative.

Exercise 1 : Order Categories, Solution b)

$$\begin{tikzcd}
x \arrow[r, "x \leq y"] \arrow[d, "F"] & y \arrow[d, "F"] \\
F(x) \arrow[r, "F(x) \leq F(y)"] & F(y)
\end{tikzcd}$$

- 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)

$$\begin{tikzcd} F(x) \arrow[d, "\phi_x \,:\, F(x) \leq G(x)"] \\ G(x) \end{tikzcd}$$

- By the definition of natural transformations, for every object x, ϕ_x is a morphism $F(x) \to G(x)$. 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:

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

Naturality from Polymorphism

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

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

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

Naturality from Polymorphism

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

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

- This is the free theorem we got for a parametrically polymorphic function `r :: [X] -> [X]` and an arbitrary function `a : A -> 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 -> G x`,

Naturality from Polymorphism

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

Naturality from Polymorphism

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

```
r .  fmap f = fmap f .  r
```

Naturality from Polymorphism

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

```
r .  fmap f = 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 -> G x`,
  - an arbitrary function `f :: A -> B`.
- Then we get the following free theorem:

```
r . fmap f = 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 -> G x`,
  - an arbitrary function `f :: A -> B`.
- Then we get the following free theorem:

```
r . fmap f = 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 $\mathbb{S}et \to \mathbb{S}et$.

Homfunctors

$$
\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "g \circ f"'] & B \arrow[d, "g"] \\
 & C
\end{tikzcd}
$$

$$
\begin{tikzcd}
\{id_A\} \arrow[r, "f \circ"] \arrow[dr, "g \circ f \circ"'] & \{f\} \arrow[d, "g \circ"] \\
 & \{g \circ f\}
\end{tikzcd}
$$

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

Homfunctors

$$
\begin{tikzcd}
A \arrow[r, "f" description] \arrow[dr, "g\circ f" description] & B \arrow[d, "g" description] \\
& C
\end{tikzcd}
$$

$$
\begin{tikzcd}
\{id_A\} \arrow[r, "f\circ" description] \arrow[dr, "g\circ f\circ" description] & \{f\} \arrow[d, "g\circ" description] \\
& \{g \circ f\}
\end{tikzcd}
$$

- For any category $\mathbb{C}$, a homset $\mathbb{C}(A, B)$ is a set of morphisms.
- We define a functor $\mathbb{C}(A, -) : \mathbb{C} \to \mathbb{S}et$:

Homfunctors

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[dr, "g \circ f"'] & B \arrow[d, "g"] \\ & C \end{tikzcd}$$

$$\begin{tikzcd} \{id_A\} \arrow[r, "f\circ"] \arrow[dr, "g \circ f \circ"'] & \{f\} \arrow[d, "g\circ"] \\ & \{g \circ f\} \end{tikzcd}$$

- For any category $\mathbb{C}$, a homset $\mathbb{C}(A, B)$ is a set of morphisms.
- We define a functor $\mathbb{C}(A, -) : \mathbb{C} \to \mathbb{S}et$:
  - $\mathbb{C}(A, -)$ maps an Object B to the Homset $\mathbb{C}(A, B)$

Homfunctors

$$
\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "g \circ f"'] & B \arrow[d, "g"] \\
& C
\end{tikzcd}
$$

$$
\begin{tikzcd}
\{id_A\} \arrow[r, "f\circ"] \arrow[dr, "g\circ f\circ"'] & \{f\} \arrow[d, "g\circ"] \\
& \{g \circ f\}
\end{tikzcd}
$$

- For any category $\mathbb{C}$, a homset $\mathbb{C}(A, B)$ is a set of morphisms.
- We define a functor $\mathbb{C}(A, -) : \mathbb{C} \to \mathbb{S}et$:
  - $\mathbb{C}(A, -)$ maps an Object B to the Homset $\mathbb{C}(A, B)$
  - A morphism $f : \mathbb{C}(B, C)$ is mapped to the morphism $f\circ : \mathbb{C}(A, B) \to \mathbb{C}(A, C)$

Homfunctors

$$
\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "g \circ f"'] & B \arrow[d, "g"] \\
& C
\end{tikzcd}
$$

$$
\begin{tikzcd}
\{id_A\} \arrow[r, "f\circ"] \arrow[dr, "g \circ f \circ"'] & \{f\} \arrow[d, "g\circ"] \\
& \{g \circ f\}
\end{tikzcd}
$$

- For any category $\mathbb{C}$, a homset $\mathbb{C}(A, B)$ is a set of morphisms.
- We define a functor $\mathbb{C}(A, -) : \mathbb{C} \to \mathbb{S}et$:
  - $\mathbb{C}(A, -)$ maps an Object B to the Homset $\mathbb{C}(A, B)$
  - A morphism $f : \mathbb{C}(B, C)$ is mapped to the morphism $f\circ : \mathbb{C}(A, B) \to \mathbb{C}(A, C)$
- Similarly, $\mathbb{C}(-, B)$ is a functor $\mathbb{C}^{op} \to \mathbb{S}et$:

Homfunctors

$$
\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "g \circ f"'] & B \arrow[d, "g"] \\
 & C
\end{tikzcd}
$$

$$
\begin{tikzcd}
\{id_A\} \arrow[r, "f\circ"] \arrow[dr, "g \circ f \circ"'] & \{f\} \arrow[d, "g\circ"] \\
 & \{g \circ f\}
\end{tikzcd}
$$

- For any category $\mathbb{C}$, a homset $\mathbb{C}(A, B)$ is a set of morphisms.
- We define a functor $\mathbb{C}(A, -) : \mathbb{C} \to \mathbb{S}et$:
  - $\mathbb{C}(A, -)$ maps an Object B to the Homset $\mathbb{C}(A, B)$
  - A morphism $f : \mathbb{C}(B, C)$ is mapped to the morphism $f \circ : \mathbb{C}(A, B) \to \mathbb{C}(A, C)$
- Similarly, $\mathbb{C}(-, B)$ is a functor $\mathbb{C}^{op} \to \mathbb{S}et$:
  - $\mathbb{C}(-, B)$ maps an Object A to the Homset $\mathbb{C}(A, B)$

Homfunctors

$$
\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "g \circ f"'] & B \arrow[d, "g"] \\
 & C
\end{tikzcd}
$$

$$
\begin{tikzcd}
\{id_A\} \arrow[r, "f\circ"] \arrow[dr, "g \circ f \circ"'] & \{f\} \arrow[d, "g\circ"] \\
 & \{g \circ f\}
\end{tikzcd}
$$

- For any category $\mathbb{C}$, a homset $\mathbb{C}(A, B)$ is a set of morphisms.
- We define a functor $\mathbb{C}(A, -) : \mathbb{C} \to \mathbb{S}et$:
  - $\mathbb{C}(A, -)$ maps an Object B to the Homset $\mathbb{C}(A, B)$
  - A morphism $f : \mathbb{C}(B, C)$ is mapped to the morphism $f\circ : \mathbb{C}(A, B) \to \mathbb{C}(A, C)$
- Similarly, $\mathbb{C}(-, B)$ is a functor $\mathbb{C}^{op} \to \mathbb{S}et$:
  - $\mathbb{C}(-, B)$ maps an Object A to the Homset $\mathbb{C}(A, B)$
  - A morphism $f : \mathbb{C}(B, A)$ is mapped to the morphism $\circ f : \mathbb{C}(A, C) \to \mathbb{C}(B, C)$

Functor Categories

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

Functor Categories

- For any $\mathbb{C}$, $\mathbb{D}$, the collection of functors $\mathbb{C} \to \mathbb{D}$ 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 $\mathbb{C} \to \mathbb{D}$ 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 $F \to G$.

The Yoneda Embedding

- Remember that the goal is finding out everything about an object A through its relations to other objects.

$$
\begin{tikzcd}
A \arrow[r, "f \,\in\, \mathbb{C}(A{,}B)"] \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"'] & B \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"] \\
\mathbb{C}(A, -) \arrow[r, "\mathcal{Y}(f)"'] & \mathbb{C}(B, -)
\end{tikzcd}
$$

The Yoneda Embedding

- Remember that the goal is finding out everything about an object A through its relations to other objects.
- So we want to describe an object through a collection of its homsets.

$$\begin{tikzcd}
A \arrow[r, "f \,\in\, \mathbb{C}(A{,}B)"] \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"'] & B \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"] \\
\mathbb{C}(A, -) \arrow[r, "\mathcal{Y}(f)"'] & \mathbb{C}(B, -)
\end{tikzcd}$$

The Yoneda Embedding

$$
\begin{tikzcd}
A \arrow[r, "f \,\in\, \mathbb{C}(A{,}B)"] \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"'] & B \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"] \\
\mathbb{C}(A, -) \arrow[r, "\mathcal{Y}(f)"'] & \mathbb{C}(B, -)
\end{tikzcd}
$$

- Remember that the goal is finding out everything about an object A through its relations to other objects.
- So we want to describe an object through a collection of its homsets.
- Formally, we want a bijective functor

$$
\begin{aligned}
\mathcal{Y} : \mathbb{C} &\to \mathbb{S}et^{\mathbb{C}} \\
A &\mapsto \mathbb{C}(A, -)
\end{aligned}
$$

The Yoneda Embedding

$$\begin{tikzcd} A \arrow[r, "f \,\in\, \mathbb{C}(A{,}B)"] \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"'] & B \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"] \\ \mathbb{C}(A,-) \arrow[r, "\mathcal{Y}(f)"'] & \mathbb{C}(B,-) \end{tikzcd}$$

- Remember that the goal is finding out everything about an object A through its relations to other objects.
- So we want to describe an object through a collection of its homsets.
- Formally, we want a bijective functor

$$\begin{aligned} \mathcal{Y} : \mathbb{C} &\to \mathbb{S}et^{\mathbb{C}} \\ A &\mapsto \mathbb{C}(A,-) \end{aligned}$$

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

The Yoneda Embedding

$$\begin{tikzcd}
A \arrow[rr, "f \in \mathbb{C}(A{,}B)"] \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"'] & & B \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"] \\
\mathbb{C}(A, -) \arrow[rr, "\mathcal{Y}(f)"'] & & \mathbb{C}(B, -)
\end{tikzcd}$$

- Remember that the goal is finding out everything about an object A through its relations to other objects.
- So we want to describe an object through a collection of its homsets.
- Formally, we want a bijective functor

$$\begin{aligned} \mathcal{Y} : \mathbb{C} &\to \mathbb{S}et^{\mathbb{C}} \\ A &\mapsto \mathbb{C}(A, -) \end{aligned}$$

- 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 $\mathbb{S}et^{\mathbb{C}}$.

The Yoneda Embedding

```tikzcd
A \arrow[r, "f \in \mathbb{C}(A{,}B)"] \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"'] & B \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"] \\
\mathbb{C}(A,-) \arrow[r, "\mathcal{Y}(f)"'] & \mathbb{C}(B,-)
```

- Remember that the goal is finding out everything about an object A through its relations to other objects.
- So we want to describe an object through a collection of its homsets.
- Formally, we want a bijective functor

$$\begin{aligned} \mathcal{Y} : \mathbb{C} &\to \mathbb{S}et^{\mathbb{C}} \\ A &\mapsto \mathbb{C}(A,-) \end{aligned}$$

- 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 $\mathbb{S}et^{\mathbb{C}}$.
- Therefore, $\mathcal{Y}(f)$ has to be a natural transformation between $\mathbb{C}(A,-)$ and $\mathbb{C}(B,-)$

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 <u>any</u> Functor $F : \mathbb{C} \to \mathbb{S}et$.

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 <u>any</u> Functor $F : \mathbb{C} \to \mathbb{S}et$.
- 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 <u>any</u> Functor $F : \mathbb{C} \to \mathbb{S}et$.
- 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 <u>any</u> Functor $F : \mathbb{C} \to \mathbb{S}et$.
- Specifically, the Yoneda lemma states that:

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

  - 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 $F : \mathbb{C} \to \mathbb{S}et$.
- Specifically, the Yoneda lemma states that:

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

  - 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 $\mathrm{Nat}(\mathbb{C}(A, -), F)$, we can construct 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 $F : \mathbb{C} \to \mathbb{S}et$.
- Specifically, the Yoneda lemma states that:

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

  - 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 $\mathrm{Nat}(\mathbb{C}(A, -), F)$, we can construct the set $F(A)$.
- Note that this is the *covariant* version of the Yoneda lemma. The lemma is sometimes stated equivalently in terms of the *contravariant homfunctor* $\mathbb{C}(-, A)$.

Constructing the bijection

$$
\begin{tikzcd}
A \arrow[r, "f"] & B \\
\mathbb{C}(A,A) \arrow[r, "\mathbb{C}(A,f)=f\circ"] \arrow[d, "\phi_A"] & \mathbb{C}(A,B) \arrow[d, "\phi_B"] \\
F(A) \arrow[r, "F(f)"] & F(B)
\end{tikzcd}
$$

- Let $\phi \in \mathrm{Nat}(\mathbb{C}(A,-),F)$. Since ϕ is natural transformation, we have

$$F(f) \circ \phi_A = \phi_B \circ f \circ$$

Constructing the bijection

$$A \xrightarrow{\quad f \quad} B$$

$$\begin{tikzcd}
\mathbb{C}(A,A) \arrow[r, "\mathbb{C}(A,f)=f\circ"] \arrow[d, "\phi_A"] & \mathbb{C}(A,B) \arrow[d, "\phi_B"] \\
F(A) \arrow[r, "F(f)"] & F(B)
\end{tikzcd}$$

- Let $\phi \in \mathrm{Nat}(\mathbb{C}(A,-), F)$. Since ϕ is natural transformation, we have

$$F(f) \circ \phi_A = \phi_B \circ f \circ$$

- Remember that these functors are $\mathbb{C} \to \mathbb{S}et$.

Constructing the bijection

$$\begin{tikzcd} A \arrow[r, "f"] & B \\ \mathbb{C}(A,A) \arrow[r, "\mathbb{C}(A,f)=f\circ"] \arrow[d, "\phi_A"] & \mathbb{C}(A,B) \arrow[d, "\phi_B"] \\ F(A) \arrow[r, "F(f)"] & F(B) \end{tikzcd}$$

- Let $\phi \in \mathrm{Nat}(\mathbb{C}(A,-), F)$. Since ϕ is natural transformation, we have

$$F(f) \circ \phi_A = \phi_B \circ f \circ$$

- Remember that these functors are $\mathbb{C} \to \mathbb{S}et$.
- This means our morphisms are just regular set functions.

Constructing the bijection

$$\begin{tikzcd}
A \arrow[r, "f"] & B \\
id_A \arrow[r, "\mathbb{C}(A,f)=f\circ"] \arrow[d, mapsto, "\phi_A"] & f \arrow[d, mapsto, "\phi_B"] \\
u \in F(A) \arrow[r, "F(f)"] & \phi_B(f) = F(f)(u)
\end{tikzcd}$$

- If we apply these functions to the identity morphism id_A, we get:

$$\phi_B(f \circ id_A) = F(f)(\phi_A(id_A))$$

$$\phi_B(f) = F(f)(\phi_A(id_A)) := F(f)(u)$$

Cayley's Theorem

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

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 $f(e)$

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 $M \to M$.

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.

$$\begin{tikzcd}
* \arrow[r, "m \in \mathbb{M}(*,*)"] \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"'] & * \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"] \\
\mathbb{M}(*,-) \arrow[r, "\mathcal{Y}(m)=m\circ"'] & \mathbb{M}(*,-)
\end{tikzcd}$$

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 $\mathbb{M}(*, -)$.

$$\begin{tikzcd}
* \arrow[rr, "m \,\in\, \mathbb{M}(*,*)"] \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"'] & & * \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"] \\
\mathbb{M}(*, -) \arrow[rr, "\mathcal{Y}(m) = m\circ"'] & & \mathbb{M}(*, -)
\end{tikzcd}$$

Exercise 2 - Cayley's Theorem for Monoids

$$\begin{tikzcd}
* \arrow[r, "m \in \mathbb{M}(*,*)"] \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"'] & * \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"] \\
\mathbb{M}(*,-) \arrow[r, "\mathcal{Y}(m)=m\circ"'] & \mathbb{M}(*,-)
\end{tikzcd}$$

- 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

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

Exercise 2 - Cayley's Theorem for Monoids

$$
\begin{tikzcd}[column sep=large, row sep=large]
* \arrow[r, "m \,\in\, \mathbb{M}(*,*)"] \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"'] & * \arrow[d, dotted, leftrightarrow, "\mathcal{Y}"] \\
\mathbb{M}(*,-) \arrow[r, "\mathcal{Y}(m)=m\circ"'] & \mathbb{M}(*,-)
\end{tikzcd}
$$

- 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

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

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

$\square$

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

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

- `s` and `t` are assumed to be composite types containing values of types `a` and `b` respectively.

Adapters

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

- `s` and `t` are assumed to be composite types containing values of types `a` and `b` respectively.
- Categorically, this translates to a pair of morphisms $\mathbb{C}(S, A) \times \mathbb{C}(B, T)$.

Adapters

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

- `s` and `t` are assumed to be composite types containing values of types `a` and `b` respectively.
- Categorically, this translates to a pair of morphisms $\mathbb{C}(S, A) \times \mathbb{C}(B, T)$.
- This is a morphism in the corresponding product category $(\mathbb{C}^{op} \times \mathbb{C})((A, B), (S, T))$

Adapters

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

- `s` and `t` are assumed to be composite types containing values of types `a` and `b` respectively.
- Categorically, this translates to a pair of morphisms $\mathbb{C}(S, A) \times \mathbb{C}(B, T)$.
- This is a morphism in the corresponding product category $(\mathbb{C}^{op} \times \mathbb{C})((A, B), (S, T))$
- We can compose adapters with matching types and define an identity adapter `Adapter id id`.

Adapters

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

- `s` and `t` are assumed to be composite types containing values of types `a` and `b` respectively.
- Categorically, this translates to a pair of morphisms $\mathbb{C}(S, A) \times \mathbb{C}(B, T)$.
- This is a morphism in the corresponding product category $(\mathbb{C}^{op} \times \mathbb{C})((A, B), (S, T))$
- We can compose adapters with matching types and define an identity adapter `Adapter id id`.
- This lets us equivalently view a category $\mathbb{C}^{op} \times \mathbb{C}$ as the category $\mathbb{A}da$ of adapters of objects of $\mathbb{C}$.

Profunctors

- A *profunctor* is a functor $\mathbb{C}^{op} \times \mathbb{C} \to \mathbb{S}et$.

Profunctors

- A *profunctor* is a functor $\mathbb{C}^{op} \times \mathbb{C} \to \mathbb{S}et$.
- As Haskell code:

```
class Profunctor p where
  dimap :: (c -> a) -> (b -> d) -> p a b -> p c d
```

Profunctors

- A *profunctor* is a functor $\mathbb{C}^{op} \times \mathbb{C} \to \mathbb{S}et$.
- As Haskell code:

```haskell
class Profunctor p where
  dimap :: (c -> a) -> (b -> d) -> p a b -> p c d
```

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

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

Profunctors

- A *profunctor* is a functor $\mathbb{C}^{op} \times \mathbb{C} \to \mathbb{S}et$.
- As Haskell code:

```haskell
class Profunctor p where
  dimap :: (c -> a) -> (b -> d) -> p a b -> p c d
```

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

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

- We define the category $\mathbb{P}rof$ to be the functor category $\mathbb{S}et^{(\mathbb{C}^{op} \times \mathbb{C})} = \mathbb{S}et^{\mathbb{A}da}$

Functor Application as a Functor

$$\begin{tikzcd}
F \arrow[rr, "\eta"] \arrow[d, dotted, "-(A)"'] & & G \arrow[d, dotted, "-(A)"] \\
F(A) \arrow[rr, "\eta_A"'] & & G(A)
\end{tikzcd}$$

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

Functor Application as a Functor

$$\begin{tikzcd}
F \arrow[r, "\eta"] \arrow[d, dotted, "-(A)"'] & G \arrow[d, dotted, "-(A)"] \\
F(A) \arrow[r, "\eta_A"'] & G(A)
\end{tikzcd}$$

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

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
```

Profunctor Adapters

- The profunctor representation of an adapter is given by:

```haskell
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.

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 whose objects are those of $\mathbb{A}da = \mathbb{C}^{op} \times \mathbb{C}$, but whose morphisms are profunctor adapters.

Profunctor Adapters

- The profunctor representation of an adapter is given by:

```haskell
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 whose objects are those of $\mathbb{A}da = \mathbb{C}^{op} \times \mathbb{C}$, but whose morphisms are profunctor adapters.
- The homsets in $\mathbb{A}daP$ are defined as:

$$\mathbb{A}daP((A, B), (S, T)) = \mathbb{S}et^{\mathbb{P}rof}(-(A, B), -(S, T))$$

Equivalence of the two representations

- The Yoneda lemma tells us that for any functor F:

$$F(A) \simeq \mathbb{S}et^{\mathbb{C}}(\mathbb{C}(A, -), F)$$

Equivalence of the two representations

- The Yoneda lemma tells us that for any functor F:

$$F(A) \simeq \mathbb{S}et^{\mathbb{C}}(\mathbb{C}(A,-), F)$$

- If we remove the reference to any specific functor, we get:

$$-(A) \simeq \mathbb{S}et^{\mathbb{C}}(\mathbb{C}(A,=), -)$$

Equivalence of the two representations

- The Yoneda lemma tells us that for any functor F:

$$F(A) \simeq \mathbb{S}et^{\mathbb{C}}(\mathbb{C}(A,-),F)$$

- If we remove the reference to any specific functor, we get:

$$-(A) \simeq \mathbb{S}et^{\mathbb{C}}(\mathbb{C}(A,=),-)$$

- So we have:

$$\begin{aligned}
&\mathbb{A}daP((A,B),(S,T)) \\
= \ & \mathbb{S}et^{\mathbb{P}rof}(-(A,B),-(S,T)) \\
\simeq \ & \mathbb{S}et^{\mathbb{P}rof}(\mathbb{P}rof(\mathbb{A}da((A,B),=),-),\mathbb{P}rof(\mathbb{A}da((S,T),=),-))
\end{aligned}$$

Equivalence of the two representations

$$
\begin{aligned}
& \mathbb{A}daP((A,B),(S,T)) \\
\simeq\ & \mathbb{S}et^{\mathbb{P}rof}(\mathbb{P}rof(\mathbb{A}da((A,B),=),-),\mathbb{P}rof(\mathbb{A}da((S,T),=),-))
\end{aligned}
$$

- Applying the Yoneda embedding of the functor category $\mathbb{P}rof = \mathbb{S}et^{\mathbb{A}da}$ gives:

$$
\begin{aligned}
& \mathbb{S}et^{\mathbb{P}rof}(\mathbb{P}rof(\mathbb{A}da((A,B),=),-),\mathbb{P}rof(\mathbb{A}da((S,T),=),-)) \\
\simeq\ & \mathbb{P}rof(\mathbb{A}da((A,B),=),\mathbb{A}da((S,T),=))
\end{aligned}
$$

- Applying the Yoneda embedding again, this time of the category $\mathbb{A}da$, gives:

$$
\begin{aligned}
& \mathbb{P}rof(\mathbb{A}da((A,B),=),\mathbb{A}da((S,T),=)) \\
=\ & \mathbb{S}et^{\mathbb{A}da}(\mathbb{A}da((A,B),=),\mathbb{A}da((S,T),=)) \\
\simeq\ & \mathbb{A}da((A,B),(S,T))
\end{aligned}
$$

Summary

- The Yoneda lemma is a fundamental result in category.

Summary

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

Summary

- The Yoneda lemma is a fundamental result in category.
- The lemma states that the set of natural transformations between a homfunctor $\mathbb{C}(A, -)$ and an arbitrary functor $F : \mathbb{C} \to \mathbb{S}et$ 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.
- The lemma states that the set of natural transformations between a homfunctor $\mathbb{C}(A, -)$ and an arbitrary functor $F : \mathbb{C} \to \mathbb{S}et$ 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 $\mathbb{C}$.
- Using the Yoneda embedding twice, we have shown the equivalence of adapters and profunctor adapters.

Summary

- The Yoneda lemma is a fundamental result in category.
- The lemma states that the set of natural transformations between a homfunctor $\mathbb{C}(A, -)$ and an arbitrary functor $F : \mathbb{C} \to \mathbb{S}et$ 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 $\mathbb{C}$.
- Using the Yoneda embedding twice, we have shown the equivalence of adapters and profunctor adapters.
- Similar techniques can be used to show the equivalence of any optic and its profunctor representation.