Just Enough Category Theory for Haskell (part 1)

Heejong Ahn
(github.com/heejongahn)

Tabel of Contents

Purpose and Focus

Purpose

- “Do I have to read this thick math book with stubborn, dull cover thoroughly, just to write a program in Haskell?”
- **You don’t**.
- We’ve already gone through 5 weeks of exercises without any book!
- Yet, as Haskell borrowed a lot of its core concepts from category theory, some knowledge on it might help you a lot.

Focus

- I’ll try to **avoid** analogues, which is often used to give an *intuition*.
- That’s because sometimes those keep you from getting actual, clear idea about a concept.
- Instead, I’d rather stick to mathematical terms and laws.

Disclaimer

- I’ve never been excellent in math.
- Haven’t been really passionate about it, neither.
- Though I’ve put a huge amount of time, there will be lots of unclear, even wrong spots throughout the slides.
- If you find any, **please feel free to interrupt and correct me** ☺

Category

Category

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

Category is just a collection which has:

- A collection of objects
- A collection of morphisms
- A notion of composition

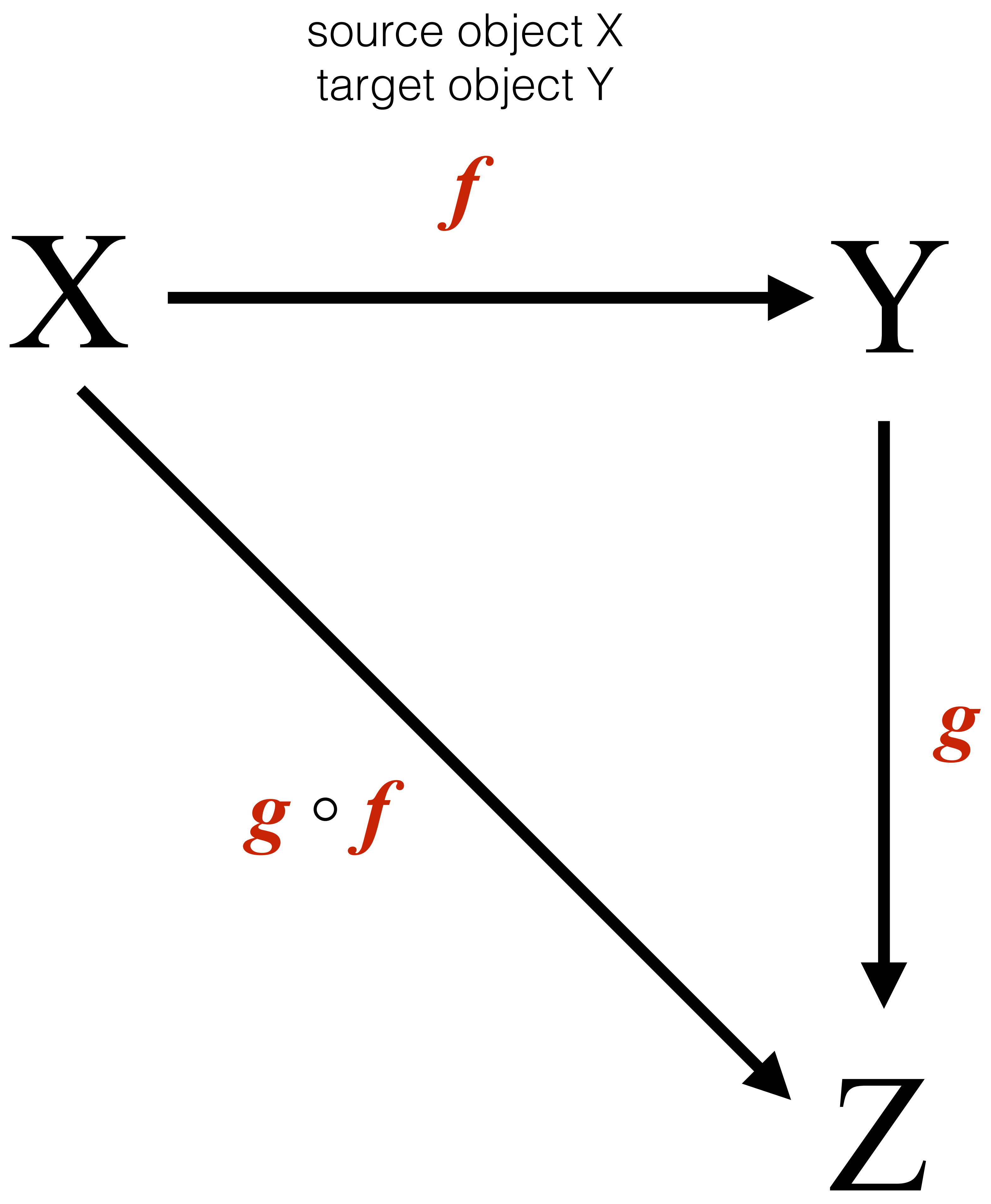
Category

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

Category is just a collection which has:

- A collection of ***objects***
- A collection of morphisms
- A notion of composition

Category



Category is just a collection which has:

- A collection of objects
- A collection of ***morphisms***
- A notion of composition

Category

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

Category is just a collection which has:

- A collection of objects
- A collection of morphisms
- A notion of ***composition***

Category Example

- Category of sets(**Set)** is a category.
- **Set** has a collection of objects, which is, well, all sets.
- **Set** has a collection of morphisms, which is a collection of functions from a set to another set.
- **Set** has a notion of composition, which is just a function composition.

Category Example (cont'd)

- Category of groups(**Grp)** is also a category.
- **Grp** has a collection of objects, which is, again, all groups.
- **Grp** has a collection of morphisms, which is a collection of functions which preserves group operation (the group homomorphism)
- **Grp** has a notion of composition, which is just a function composition.

Category Laws

Categories must meet these requirements :

- Composition of mophisms should be associative $f \circ (g \circ h) = (f \circ g) \circ h$
- For every object in a category, there must be an identity morphism

Hask

- **Hask**, the category of Haskell types and functions, is our main topic.
- Speaking of category, we’ve heard that every category must have…
- *Objects*, *morphisms*, and *composition*!

Hask (cont'd)

- Objects : Haskell **types**
- Morphisms : Haskell **functions**
- Composition : **(.)**
- We know haskell functions are associative, and we have `id`.
- On top of that, we *define* two functions `f` and `g` as the same morphism if `f x` = `g x` for all `x`. *(Why? Is it really needed?)*

Functor

Functor

- “In mathematics, a functor is a type of mapping between categories which is applied in category theory. Functors can be thought of as homomorphisms between categories.” (Wikipedia)
- Sorry, what?

Functor (cont'd)

- "In mathematics, a functor is a type of ***mapping between categories*** which is applied in category theory. Functors can be thought of as ***homomorphisms between categories***."

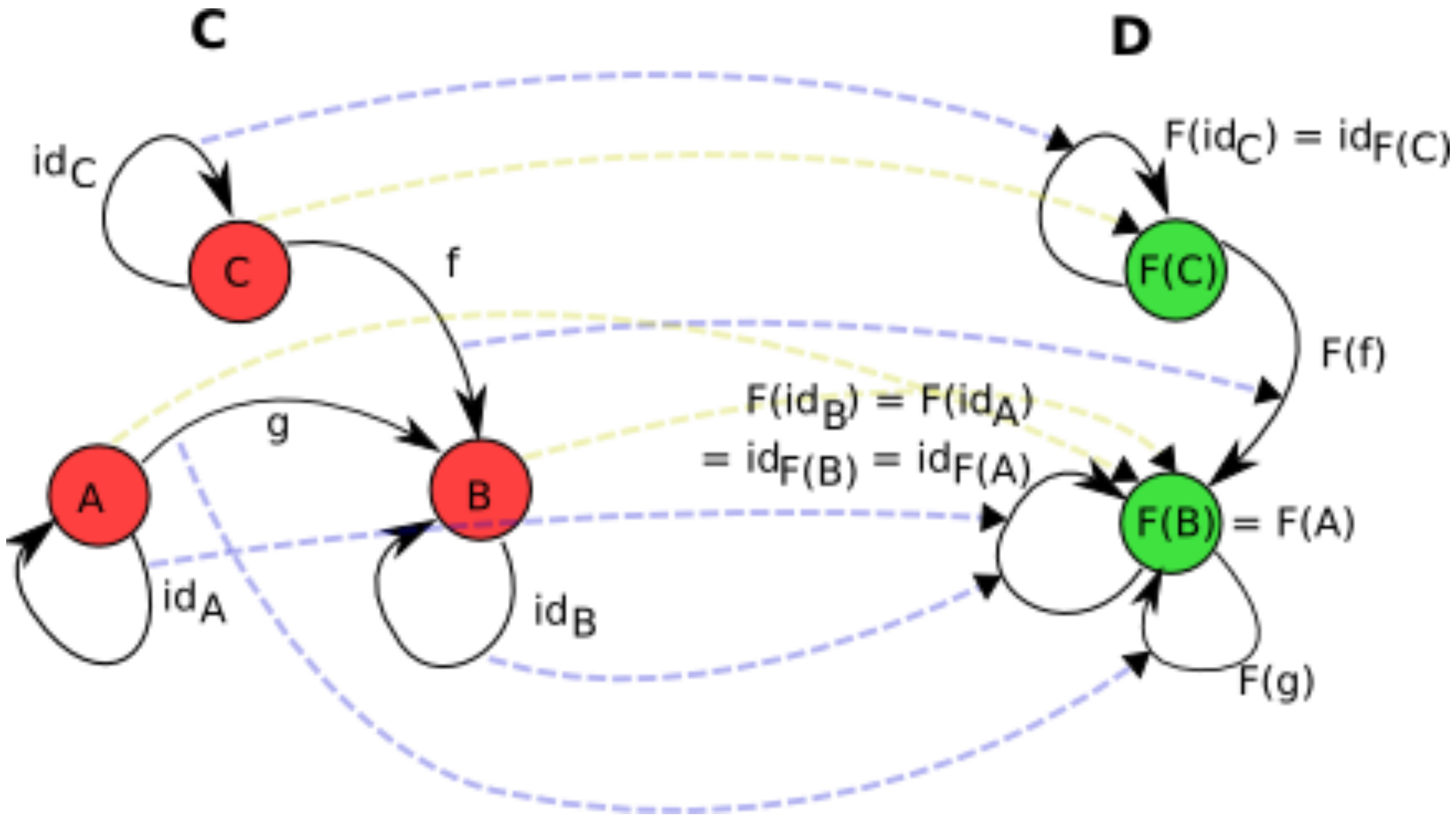
- To map a category to another category, we have to map…
- Objects, Morphisms, and Composition!
- Composition is mostly trivial for this case, so we won't focus on it.

Functor Laws

Functors must meet these requirements :

- Functor must map identity of an object to that of a mapped object $F(id_A) = id_{F(A)}$
- Functor must distribute over morphism composition $F(f \circ g) = F(f) \circ F(g)$

Functor Example



- Diagram of a functor from category C to category D.
- You can see:
  - Object mapping (yellow)
  - Morphism mapping (purple)

Functor in **Hask**

```
class Functor (f :: * -> *) where
  fmap :: (a -> b) -> f a -> f b

instance Functor Maybe where
  fmap f (Just x) = Just (f x)
  fmap _ Nothing  = Nothing
```

- What does `Maybe` do?
  - It turns a **Hask** object(type), T, to another object, `Maybe T`.

Functor in **Hask** (cont'd)

```
class Functor (f :: * -> *) where
  fmap :: (a -> b) -> f a -> f b

instance Functor Maybe where
  fmap f (Just x) = Just (f x)
  fmap _ Nothing  = Nothing
```

- What does `fmap` do, when applied to a `Maybe` instance?
  - It turns a **Hask** morphism(function), with type (`A -> B`), to another morphism, with type (`Maybe A -> Maybe B`).

Functor in **Hask** (cont'd)

- So as we have
  - **Type constructor**, which maps objects between categories
  - **`fmap`**, which maps morphisms between categories
- `Maybe` is an instance of `Functor` - it maps **Hask** to its subcategory, a category of `Maybe` types and functions defined on `Maybe` types.

Functor in **Hask** (cont'd)

- Same logic applies to any `Functor` types.
- Both Functor laws are fulfilled for any functor on **Hask**.
  - `fmap id = id`
  - `fmap (f . g) = (fmap f) . (fmap g)`

Summary

Summary

- We've talked about Hask, the category of Haskell types and functions.
- (types, functions, (.)) corresponds to (objects, morphisms, composition), respectively.
- `Functor` typeclasses works as Functor in category theory, that is, a mapping between two different categories.
- Next topics : Monoid, Applicative Functor, Monad, …

Appendix

References

- https://en.wikibooks.org/wiki/Haskell/Category_theory
- https://wiki.haskell.org/Hask
- https://en.wikipedia.org/wiki/Category_theory
- https://en.wikipedia.org/wiki/Category_(mathematics)
- https://en.wikipedia.org/wiki/Group_(mathematics)
- https://en.wikipedia.org/wiki/Group_homomorphism
- https://en.wikipedia.org/wiki/Category_of_sets
- https://en.wikipedia.org/wiki/Category_of_groups
- https://wiki.haskell.org/Typeclassopedia