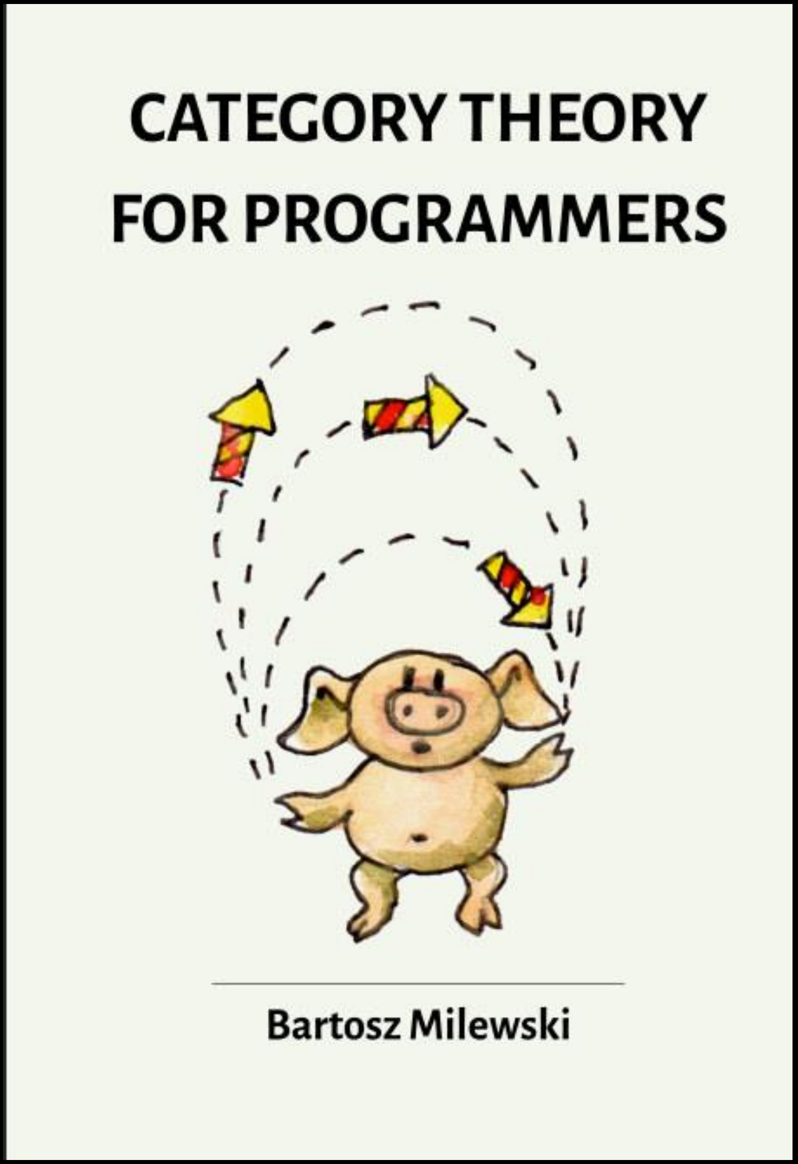
Java

Meetup

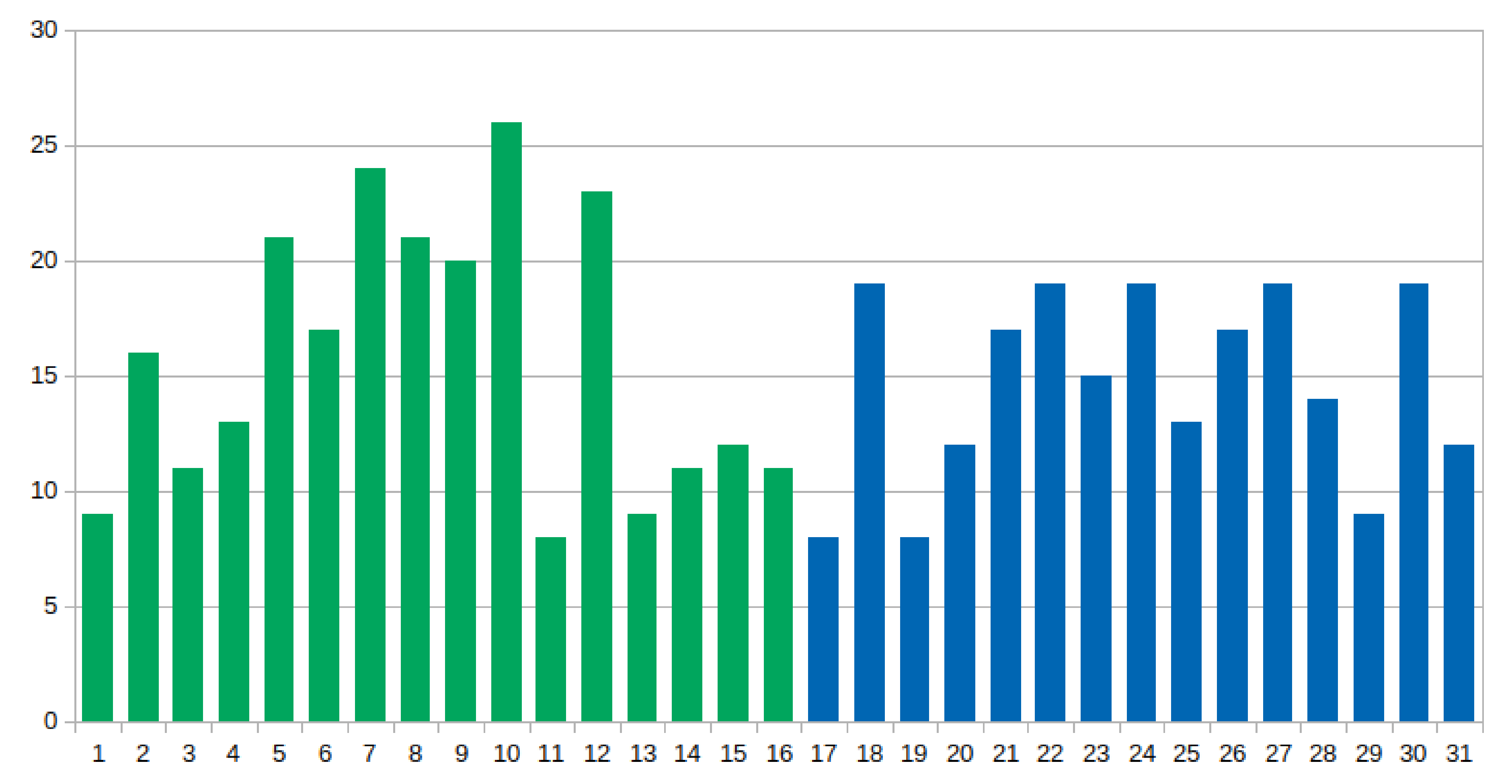
Friendly Environment Policy

Berlin Code of Conduct



Category Theory for Programmers

REVIEW


30
25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Main Concepts	Supporting Concepts
Universal Construction (technique)	
A) **Objects**	Naturality Condition
B) **Morphisms** Also known as “arrows”	Isomorphism
C) **Category** (A + B)	Hom-Set
D) **Functor** (C + B) Bifunctor, Profunctor, Contravariant Functor Product, Coproduct || Maybe, List, Reader	Hom-Functor
E) **Natural Transformation** (D + B)	Yoneda Lemma
	Yoneda Embedding

Main Concepts	Supporting Concepts
Universal Construction (technique)	
A) **Objects**	Naturality Condition
B) **Morphisms** Also known as “arrows”	Isomorphism
C) **Category** (A + B)	Hom-Set
D) **Functor** (C + B) Bifunctor, Profunctor, Contravariant Functor Product, Coproduct || Maybe, List, Reader	Hom-Functor
E) **Natural Transformation** (D + B)	Yoneda Lemma
	Yoneda Embedding

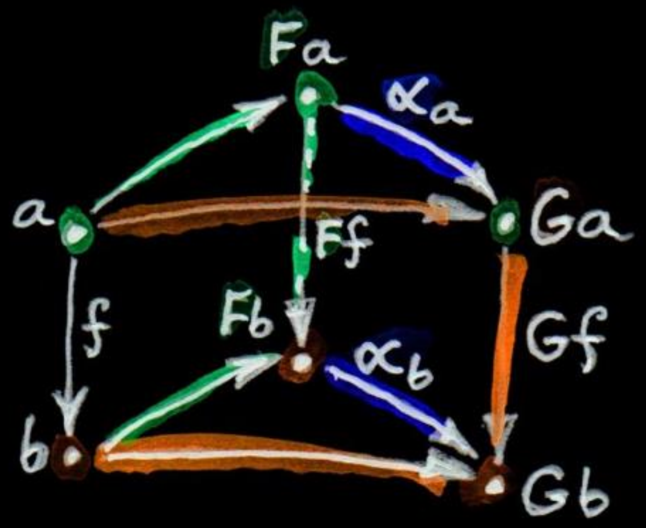
Main Concepts	Supporting Concepts
Universal Construction (technique)	
A) **Objects**	Naturality Condition
B) **Morphisms** Also known as “arrows”	(Natural) Isomorphism
C) **Category** (A + B)	Hom-Set
D) **Functor** (C + B) Bifunctor, Profunctor, Contravariant Functor Product, Coproduct || Maybe, List, Reader	Hom-Functor
E) **Natural Transformation** (D + B)	Yoneda Lemma
	Yoneda Embedding

SO FAR I've been glossing over the meaning of function types. A function type is different from other types.

Take `Integer`, for instance: It's just a set of integers. `Bool` is a two element set. But a function type $a \to b$ is more than that: it's a set of morphisms between objects a and b. A set of morphisms between two objects in any category is called a hom-set. It just so happens that in the category **Set** every hom-set is itself an object in the same category —because it is, after all, a *set*.

the category. We don't want to make artificial connections between objects. So it's *natural* to use existing connections, namely morphisms. A natural transformation is a selection of morphisms: for every object a, it picks one morphism from Fa to Ga. If we call the natural transformation α, this morphism is called the *component* of α at a, or α_a.

$$\alpha_a :: Fa \to Ga$$

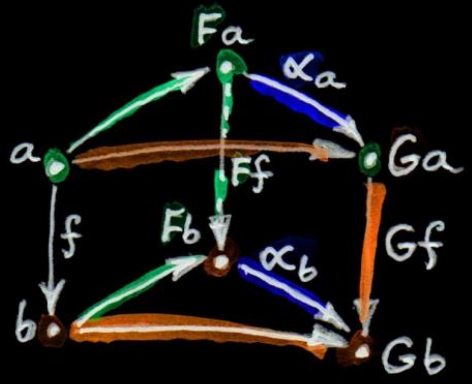


Now we have two ways of getting from Fa to Gb. To make sure that they are equal, we must impose the *naturality condition* that holds for any f:

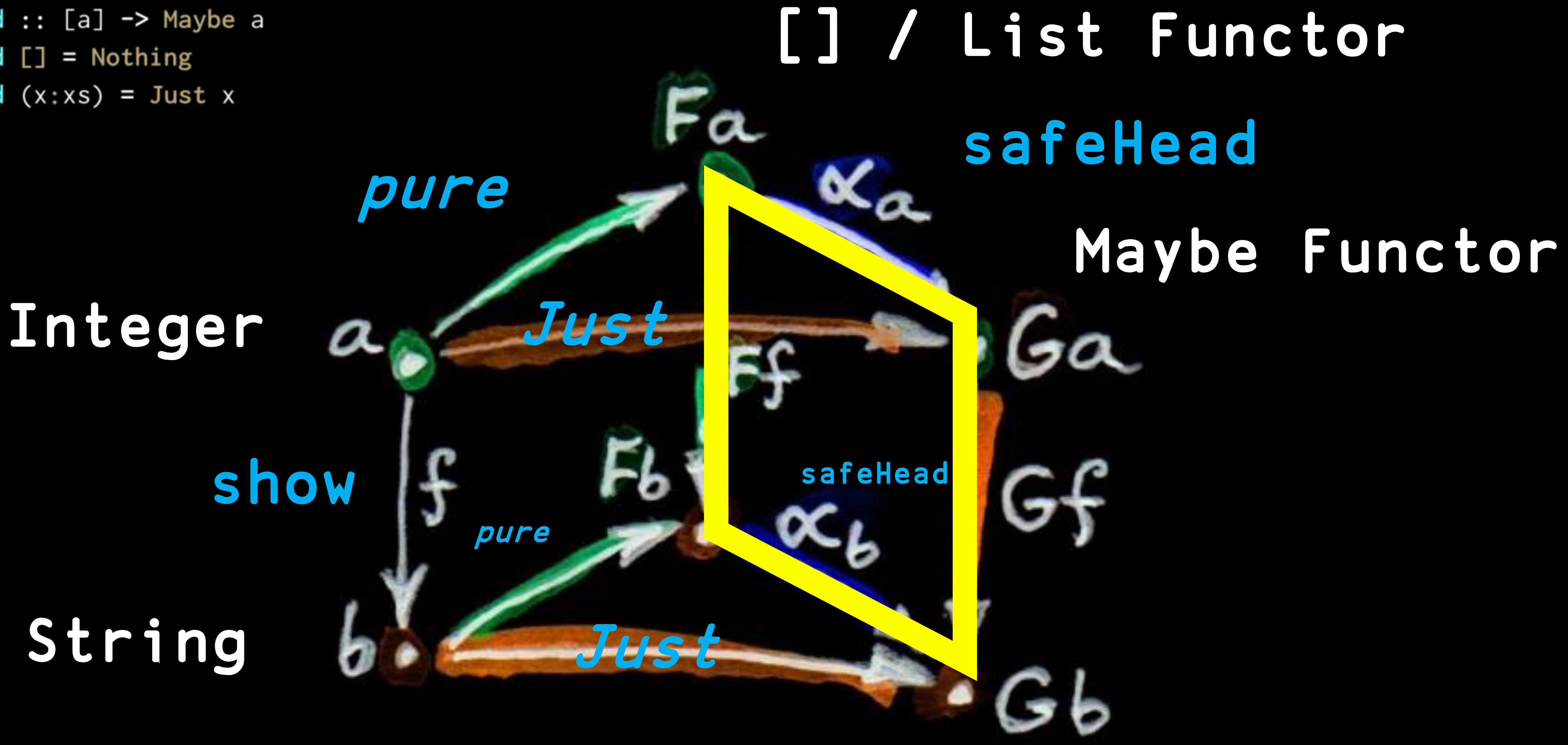
$$Gf \circ \alpha_a = \alpha_b \circ Ff$$

Let's see a few examples of natural transformations in Haskell. The first is between the list functor, and the Maybe functor. It returns the head of the list, but only if the list is non-empty:

```
safeHead :: [a] -> Maybe a
safeHead [] = Nothing
safeHead (x:xs) = Just x
```



```haskell
safeHead :: [a] -> Maybe a
safeHead [] = Nothing
safeHead (x:xs) = Just x
```



Chapter 11-16 // TODO

Java

Meetup