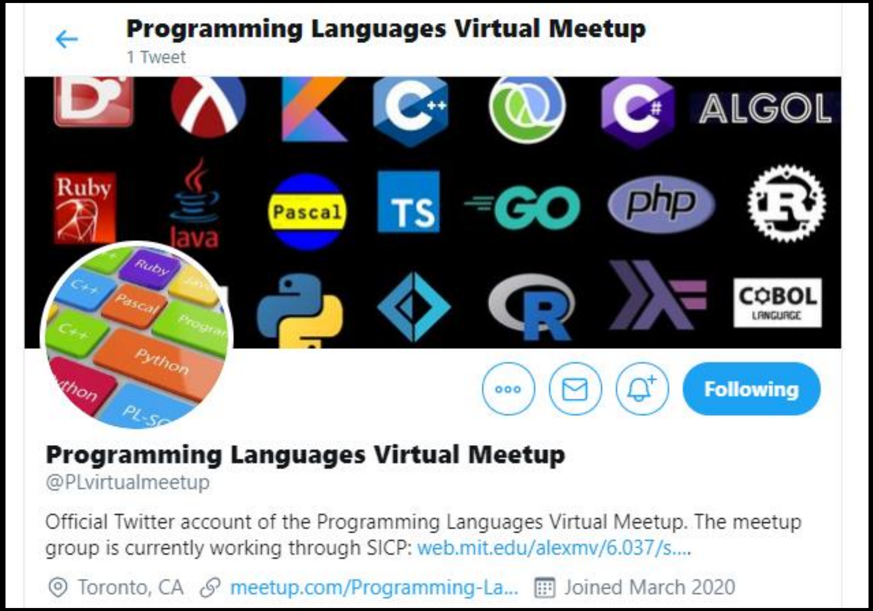
Java

Meetup

Friendly Environment Policy

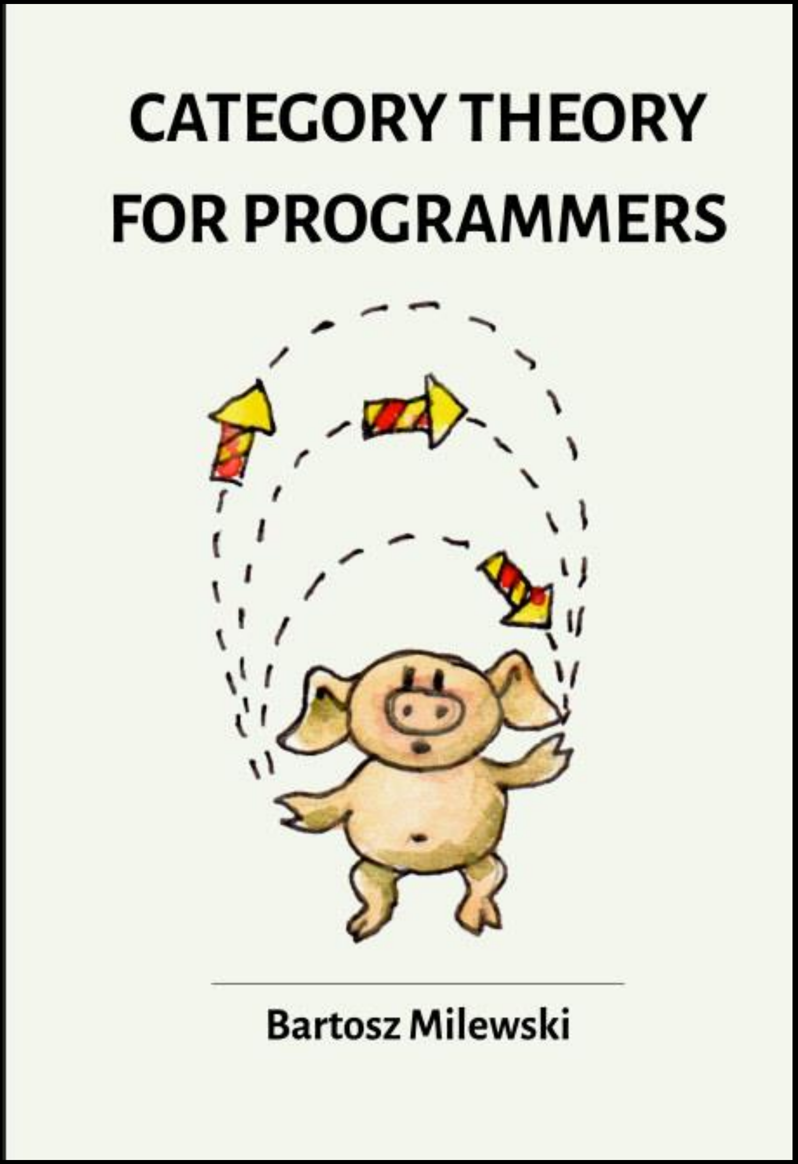
Berlin Code of Conduct


Programming Languages Virtual Meetup
1 Tweet

Programming Languages Virtual Meetup
@PLvirtualmeetup

Official Twitter account of the Programming Languages Virtual Meetup. The meetup group is currently working through SICP: web.mit.edu/alexmv/6.037/s....

Toronto, CA meetup.com/Programming-La... Joined March 2020

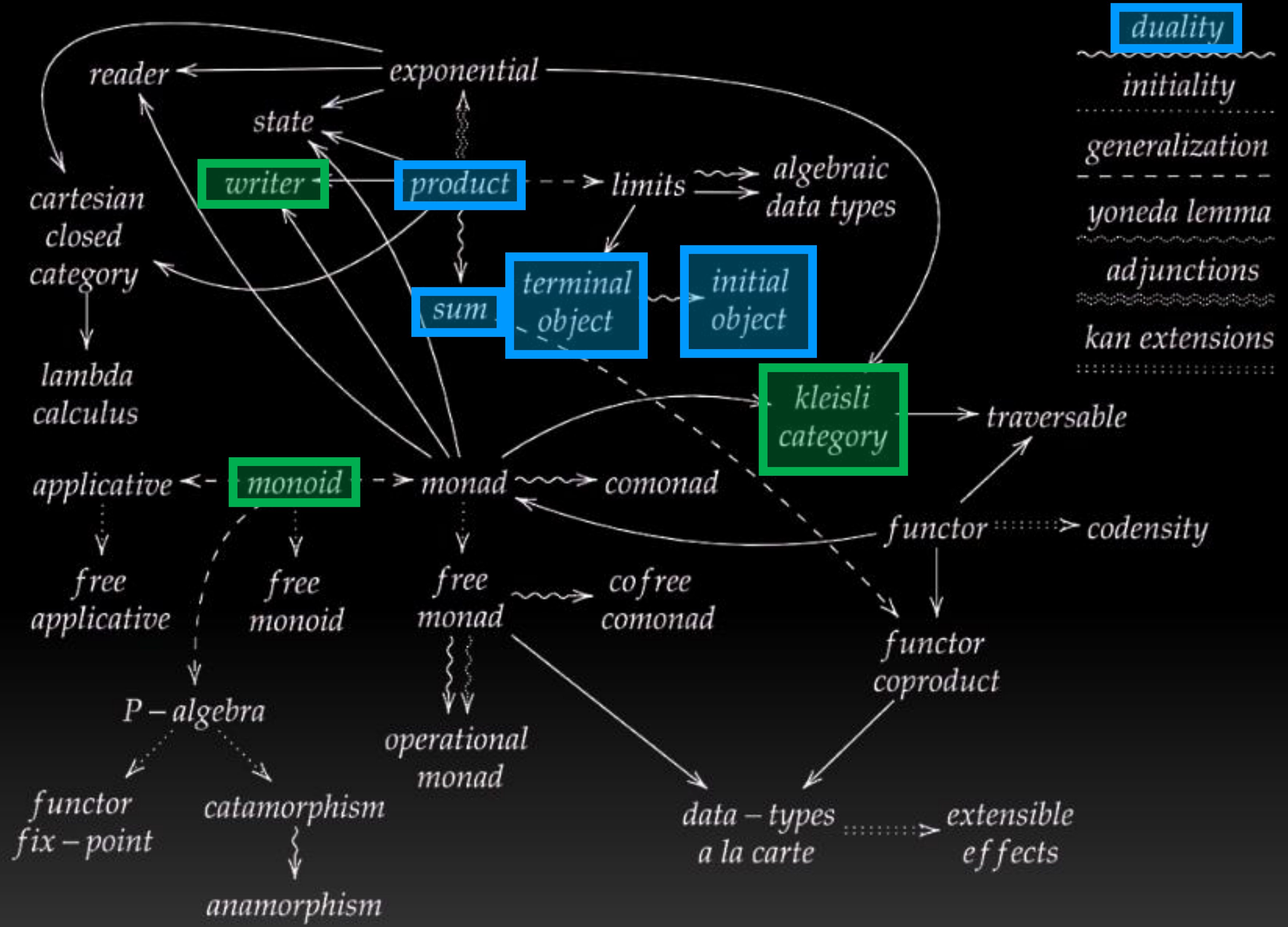


Category Theory for Programmers

Chapter 5:

Products & Coproducts

The Tools for Thought



The **initial object** is the object that has one and only one morphism going to any object in the category.

The **terminal object** is the object with one and only one morphism coming to it from any object in the category.

5.3 Duality

You can't help but to notice the symmetry between the way we defined the initial object and the terminal object. The only difference between the two was the direction of morphisms. It turns out that for any category $\mathbf{C}$ we can define the *opposite category* $\mathbf{C}^{op}$ just by reversing all the arrows.

Mathematically it means that there is a mapping from object a to object b, and there is a mapping from object b back to object a, and they are the inverse of each other. In category theory we replace mappings with morphisms. An isomorphism is an invertible morphism; or a pair of morphisms, one being the inverse of the other.

Now let’s forget about sets and define a product of two objects in any category using the same universal construction. Such a product doesn’t always exist, but when it does, it is unique up to a unique isomorphism.

When I said that the initial (terminal) object was unique up to isomorphism, I meant that any two initial (terminal) objects are isomorphic.

A **product** of two objects a and b is the object c equipped with two projections such that for any other object c' equipped with two projections there is a unique morphism m from c' to c that factorizes those projections.

A **coproduct** of two objects a and b is the object c equipped with two injections such that for any other object c' equipped with two injections there is a unique morphism m from c to c' that factorizes those injections.

1. Show that the terminal object is unique up to unique isomorphism.

What does 'unique up to isomorphism' really mean?

Many thanks to Jens Seeber for comments and corrections!

Phrases like "*Terminal objects are unique up to isomorphism*" are everywhere in category theory. In this post, I'll explain the concept of "uniqueness up to isomorphism", and its best buddy, "uniqueness up to *unique* isomorphism".

I'll also talk about the philosophy behind these concepts, and how they allow category theorists to define objects in terms of their **relationship to other objects** rather than by some internal properties. In other words, how can we avoid asking "what *is* this object?", but instead ask "how does this object *behave*?".

I will assume only a minimal background in category theory, but you should at least know what a category is, and what is meant by "objects" and "morphisms".

We'll begin with some intuition for how isomorphism mathematically means "same shape". Next, we'll make that intuition precise, before examining the example of terminal objects, which are both unique up to isomorphism, and up to *unique* isomorphism.

$$\begin{tikzcd} A \arrow[r, bend left, "\alpha"] \arrow["id_A", loop left] & B \arrow[l, bend left, "\alpha^{-1}"] \arrow["id_B", loop right] \end{tikzcd}$$

4. Implement the equivalent of Haskell `Either` as a generic type in your favorite language (other than Haskell).
5. Show that `Either` is a "better" coproduct than `int` equipped with two injections:

```
left      ← 1,⊢
right     ← 2,⊢
isLeft    ← 1=⊃
isRight   ← 2=⊃
fromLeft  ← { isLeft  ⍵: ⊃⌽⍵ ⋄ 'Fail' ⎕SIGNAL 999 }
fromRight ← { isRight ⍵: ⊃⌽⍵ ⋄ 'Fail' ⎕SIGNAL 999 }
```

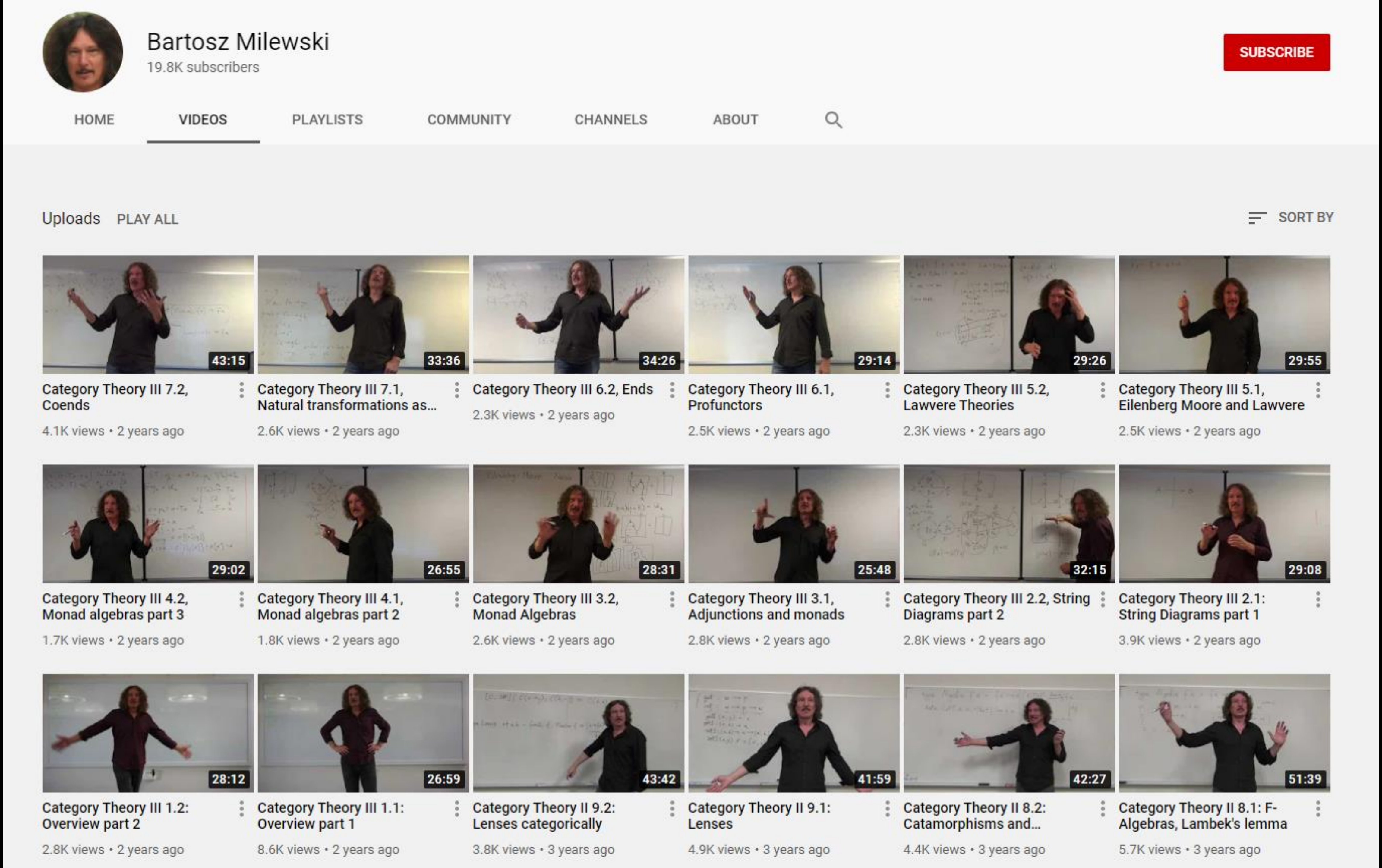
```
left      ← 1,⊢
right     ← 2,⊢
isLeft    ← 1=⊃
isRight   ← 2=⊃
fromLeft  ← { isLeft  ⍵: ⊃⌽⍵ ⋄ 'Fail' ⎕SIGNAL 999 }
fromRight ← { isRight ⍵: ⊃⌽⍵ ⋄ 'Fail' ⎕SIGNAL 999 }

l ← left  42
r ← right 1729

isLeft    l ⍝ 1
isRight   r ⍝ 1
fromLeft  l ⍝ 42
fromRight r ⍝ 1729
fromLeft  r ⍝ Fail
fromRight l ⍝ Fail
```

```
i ← ⊢
j ← 0∘≠

m ← { isLeft ⍵: fromLeft     ⍵
              ⋄ 0≠fromRight ⍵ }
```



Bartosz Milewski
19.8K subscribers

SUBSCRIBE

HOME VIDEOS PLAYLISTS COMMUNITY CHANNELS ABOUT

Uploads PLAY ALL SORT BY

43:15
Category Theory III 7.2, Coends
4.1K views • 2 years ago

33:36
Category Theory III 7.1, Natural transformations as...
2.6K views • 2 years ago

34:26
Category Theory III 6.2, Ends
2.3K views • 2 years ago

29:14
Category Theory III 6.1, Profunctors
2.5K views • 2 years ago

29:26
Category Theory III 5.2, Lawvere Theories
2.3K views • 2 years ago

29:55
Category Theory III 5.1, Eilenberg Moore and Lawvere
2.5K views • 2 years ago

29:02
Category Theory III 4.2, Monad algebras part 3
1.7K views • 2 years ago

26:55
Category Theory III 4.1, Monad algebras part 2
1.8K views • 2 years ago

28:31
Category Theory III 3.2, Monad Algebras
2.6K views • 2 years ago

25:48
Category Theory III 3.1, Adjunctions and monads
2.8K views • 2 years ago

32:15
Category Theory III 2.2, String Diagrams part 2
2.8K views • 2 years ago

29:08
Category Theory III 2.1: String Diagrams part 1
3.9K views • 2 years ago

28:12
Category Theory III 1.2: Overview part 2
2.8K views • 2 years ago

26:59
Category Theory III 1.1: Overview part 1
8.6K views • 2 years ago

43:42
Category Theory II 9.2: Lenses categorically
3.8K views • 3 years ago

41:59
Category Theory II 9.1: Lenses
4.9K views • 3 years ago

42:27
Category Theory II 8.2: Catamorphisms and...
4.4K views • 3 years ago

51:39
Category Theory II 8.1: F-Algebras, Lambek's lemma
5.7K views • 3 years ago

$$\begin{tikzcd}
 & c' \arrow[ddl, bend right, "p'"'] \arrow[ddr, bend left, "q'"] \arrow[d, dashed, "m"] & \\
 & c \arrow[dl, "p"] \arrow[dr, "q"'] & \\
a & & b
\end{tikzcd}$$

$p' = p \circ m$

$q' = q \circ m$

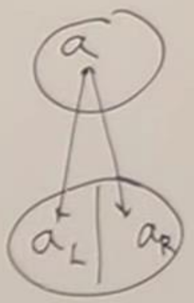
(a, b)

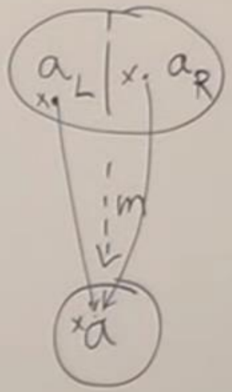
pair $\langle a, b \rangle$

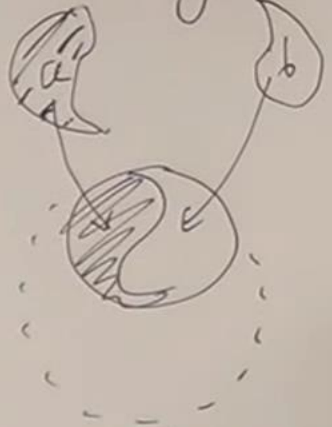
$$\begin{tikzcd}
a \arrow[dr, "i"] \arrow[ddr, bend right, "i'"'] & & b \arrow[dl, "j"'] \arrow[ddl, bend left, "j'"] \\
 & c \arrow[d, dashed, "m"] & \\
 & c' &
\end{tikzcd}$$

$i' = m \circ i$

$j' = m \circ j$







tagged union

variant

Java

Meetup