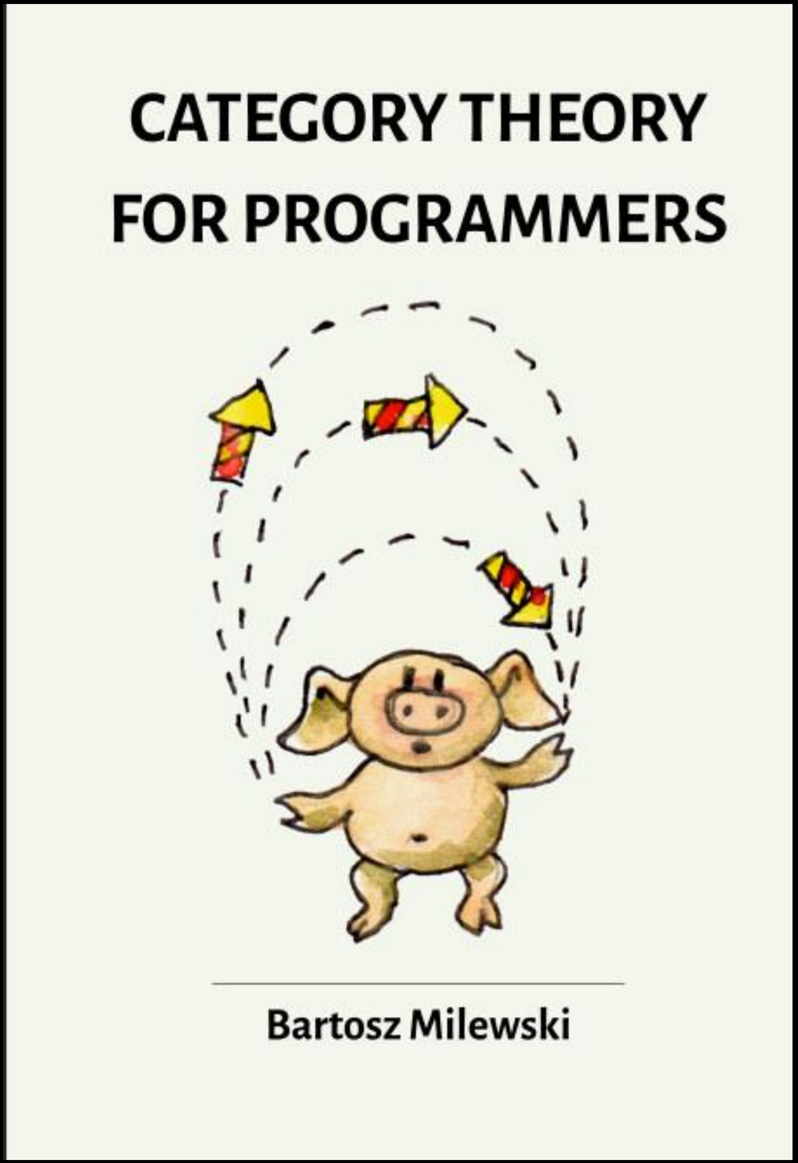
Java

Meetup

Friendly Environment Policy

Berlin Code of Conduct

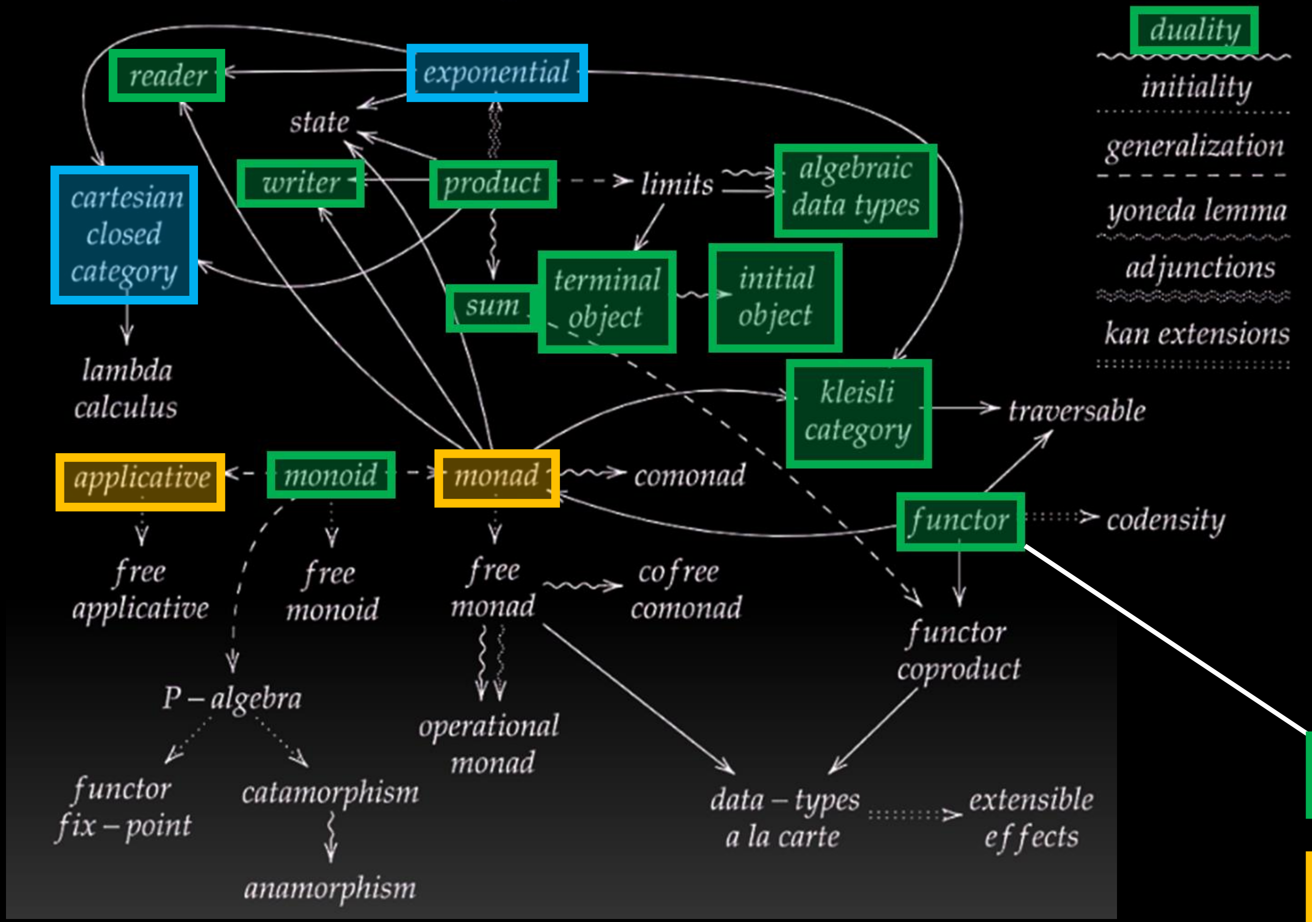
DISCORD

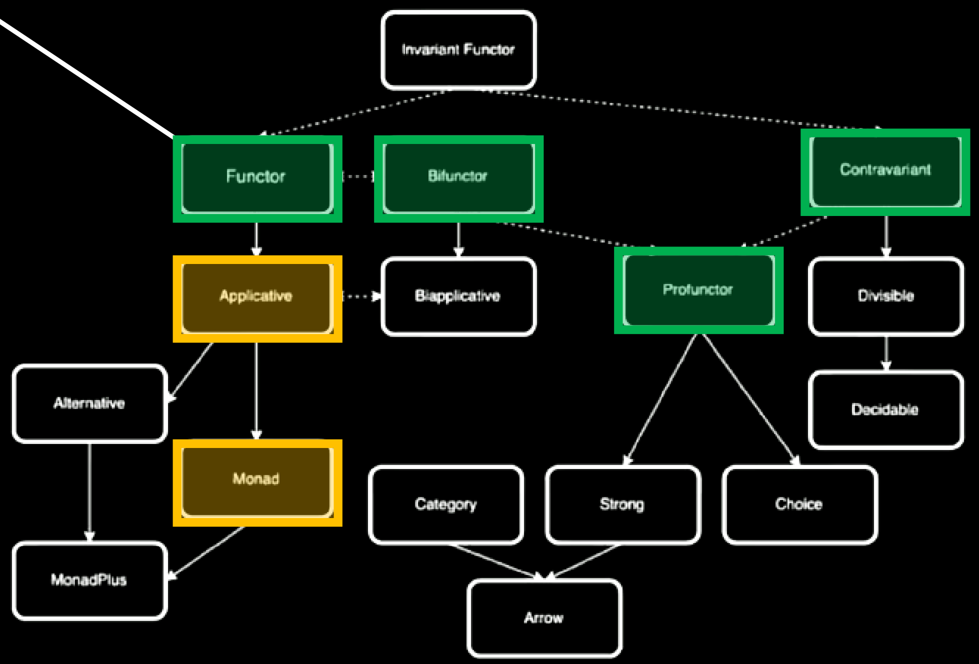


Category Theory for Programmers

Chapter 9: Function Types

The Tools for Thought





9.2 Currying

Currying is essentially built into the syntax of Haskell. A function returning a function:

```
a -> (b -> c)
```

is often thought of as a function of two variables. That's how we read the un-parenthesized signature:

```
a -> b -> c
```

Strictly speaking, a function of two variables is one that takes a pair (a product type):

```
(a, b) -> c
```

It's trivial to convert between the two representations, and the two (higher-order) functions that do it are called, unsurprisingly, `curry` and `uncurry`:

```haskell
curry :: ((a, b) -> c) -> (a -> b -> c)
curry f a b = f (a, b)
```

and

```haskell
uncurry :: (a -> b -> c) -> ((a, b) -> c)
uncurry f (a, b) = f a b
```

Meeting C++ 2019

Better
Algorithm Intuition

Conor Hoekstra (he/him)

code_report

codereport

NVIDIA RAPIDS

https://rapids.ai

Conor Hoekstra

Better Algorithm Intuition

Meeting C++ 2019

LeetCode

905. Sort Array By Parity

Easy 586 62 Favorite Share

Given an array `A` of non-negative integers, return an array consisting of all the even elements of `A`, followed by all the odd elements of `A`.

You may return any answer array that satisfies this condition.

https://leetcode.com/problems/sort-array-by-parity/

Conor Hoekstra

Better Algorithm Intuition

Meeting C++ 2019

```cpp
auto sortArrayByParity(vector<int>& A) {

    std::partition(
        std::begin(A),
        std::end(A),
        [](auto e) {
            return e % 2 == 0;
        });

    return A;
}
```

Thrust thrust::partition

Conor Hoekstra

Better Algorithm Intuition

```cpp
auto sort_array_by_parity(auto& A) {

    std::ranges::partition(A,
        [](auto e) { return e % 2 == 0; });

    return A;
}
```

```cpp
auto sort_array_by_parity(auto& A) {

    auto is_even = [](auto e) { return e % 2 == 0; };
    std::ranges::partition(A, is_even);

    return A;
}
```


Better Algorithm Intuition - Conor Hoekstra @code_report - Meeting C++ 2019
Meeting C++ 2019
[1,2,3,4,5,6]
([1,3,5],[2,4,6])
[1,3,5,2,4,6]
sortArrayByParity :: [Int] -> [Int]
sortArrayByParity = uncurry (++)
. partition odd
33
Conor Hoekstra
Better Algorithm Intuition
13:39 / 58:33

```
Prelude Data.List> partition odd [1..10]
([1,3,5,7,9],[2,4,6,8,10])

Prelude Data.List> :t (++)
(++) :: [a] -> [a] -> [a]

Prelude Data.List> :t uncurry (++)
uncurry (++) :: ([a], [a]) -> [a]
```

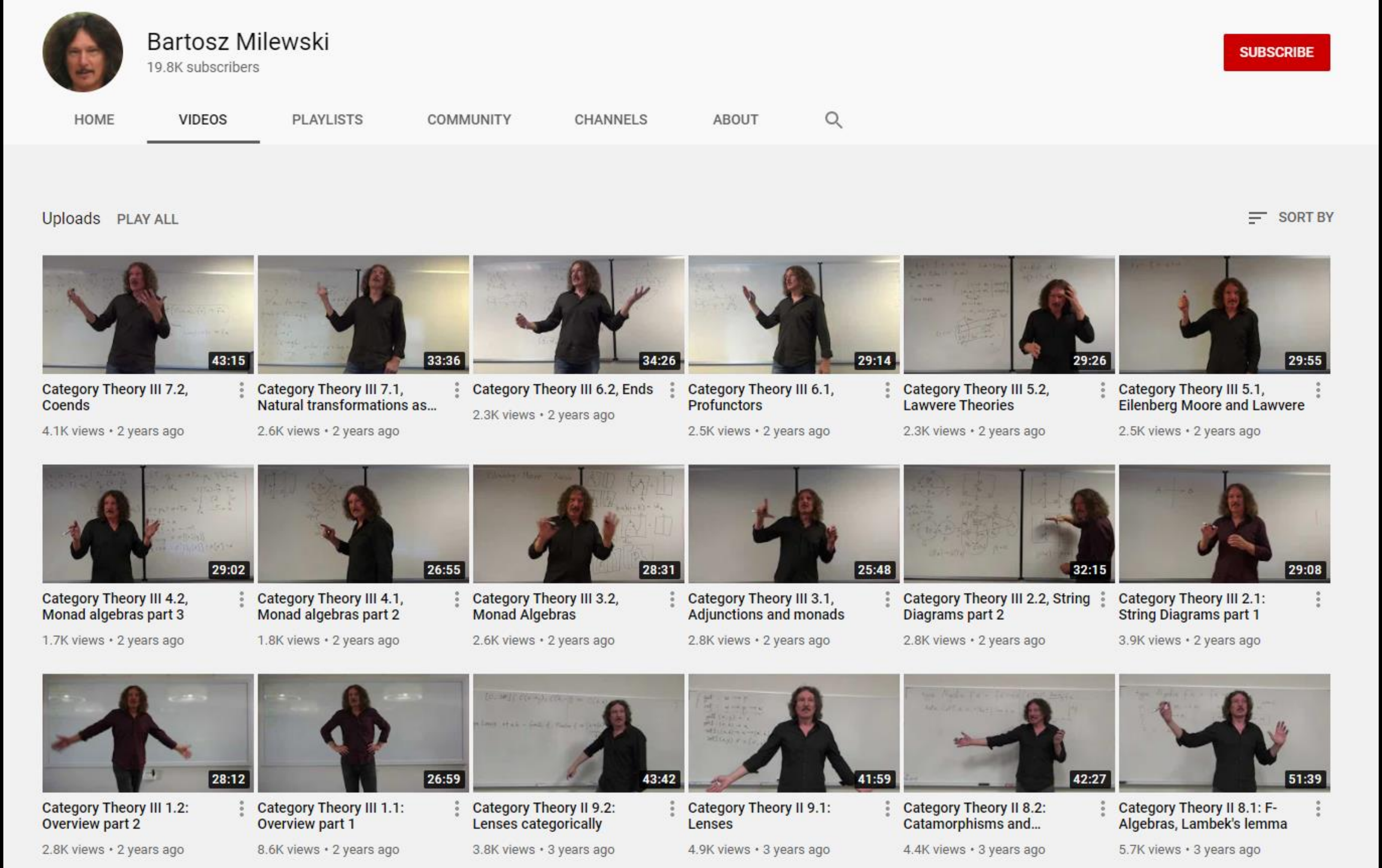
9.4 Cartesian Closed Categories

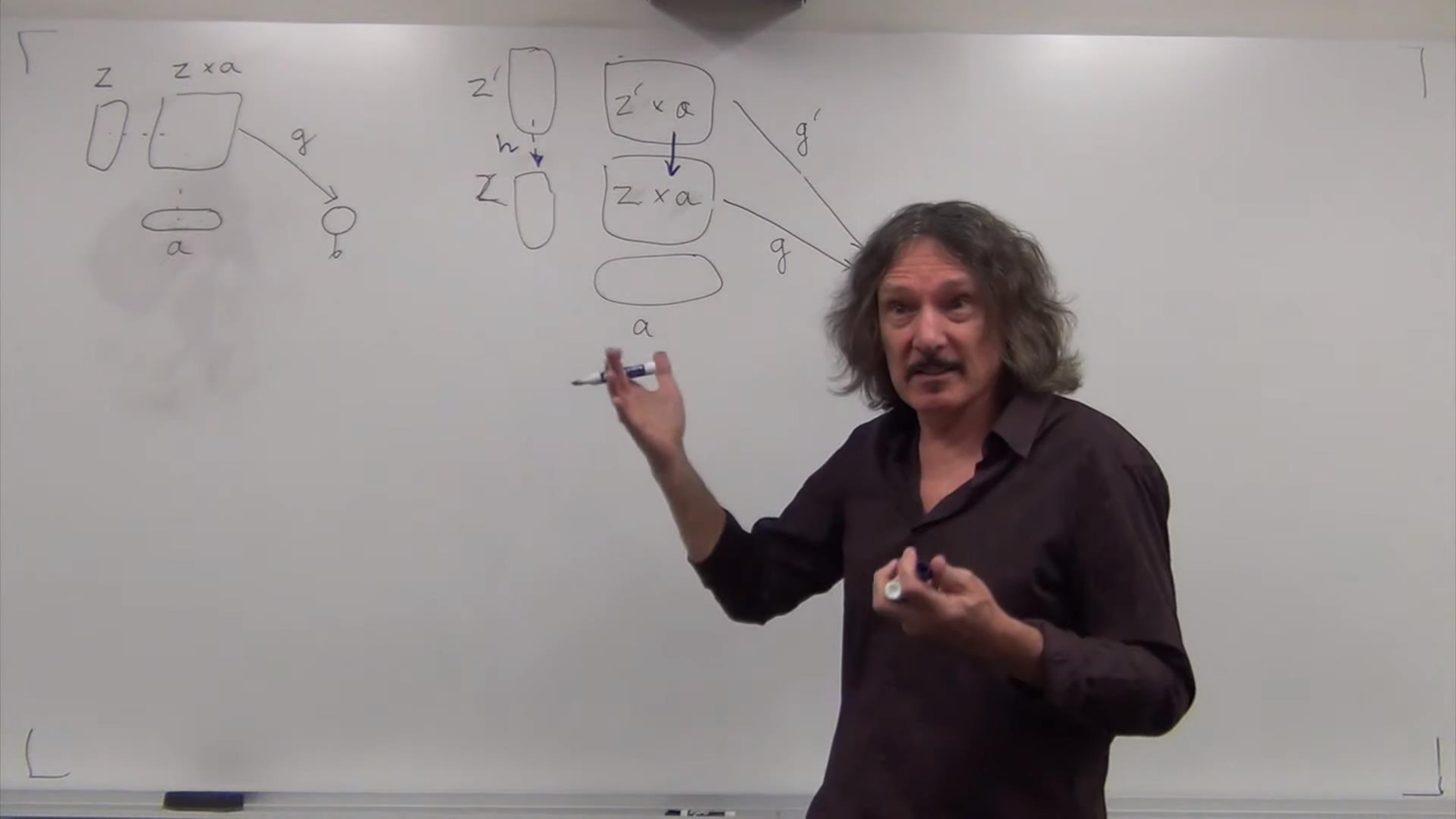
Although I will continue using the category of sets as a model for types and functions, it's worth mentioning that there is a larger family of categories that can be used for that purpose. These categories are called *Cartesian closed*, and **Set** is just one example of such a category.

A Cartesian closed category must contain:

1. The terminal object,
2. A product of any pair of objects, and
3. An exponential for any pair of objects.

Computers are not only helping mathematicians do their work — they are revolutionizing the very foundations of mathematics. The latest hot research topic in that area is called Homotopy Type Theory, and is an outgrowth of type theory. It's full of Booleans, integers, products and coproducts, function types, and so on. And, as if to dispel any doubts, the theory is being formulated in Coq and Agda. Computers are revolutionizing the world in more than one way.


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


Z
Z × a
g
a
b
Z'
Z' × a
g'
h
Z
Z × a
g
a

I needed the functoriality of the product … a product is a bifunctor … not only does it take two objects and produces a third object but does the same for morphisms.

Bartosz Milewski – Lecture 8.1

Java

Meetup