Java

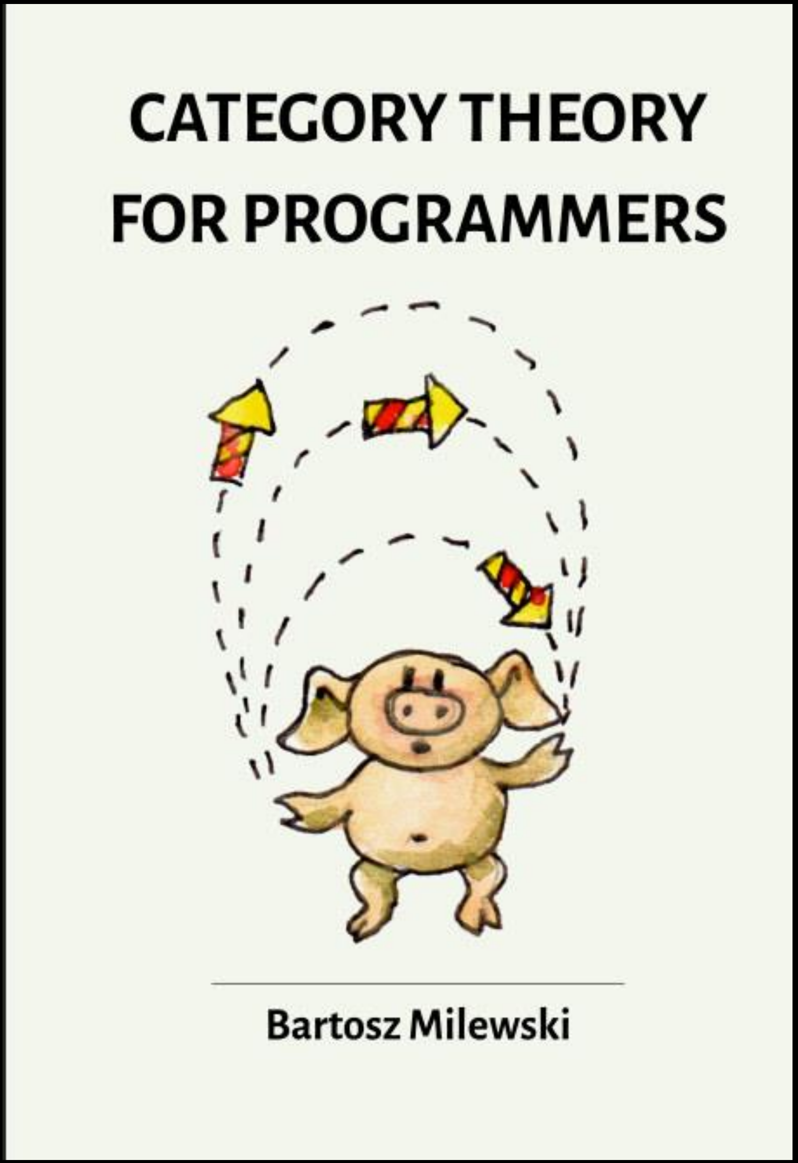
Meetup

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 18: Adjunctions

You really don’t understand adjunctions until you have seen several examples.

You really don’t understand adjunctions until you have seen several examples (because they are pretty abstract things).

You really don’t understand adjunctions until you have seen several examples (because they are pretty abstract things).

Bartosz Milewski

Minute 18 of 6.1 Examples of Adjunctions

In mathematics we have various ways of saying that one thing is like another. The strictest is equality. Two things are equal if there is no way to distinguish one from another. One can be substituted for the other in every imaginable context. For instance, did you notice that we used *equality* of morphisms every time we talked about commuting diagrams? That's because morphisms form a set (hom-set) and set elements can be compared for equality.

But equality is often too strong. There are many examples of things being the same for all intents and purposes, without actually being equal. For instance, the pair type `(Bool, Char)` is not strictly equal to `(Char, Bool)`, but we understand that they contain the same information. This concept is best captured by an *isomorphism* between two types — a morphism that's invertible. Since it's a morphism, it preserves the structure; and being "iso" means that it's part of a round trip that lands you in the same spot, no matter on which side you start. In the case of pairs, this isomorphism is called `swap`:

```haskell
swap :: (a,b) -> (b,a)
swap (a,b) = (b,a)
```

`swap` happens to be its own inverse.

So, instead of isomorphism of categories, it makes sense to consider a more general notion of *equivalence*. Two categories **C** and **D** are *equivalent* if we can find two functors going back and forth between them, whose composition (either way) is *naturally isomorphic* to the identity functor. In other words, there is a two-way natural transformation between the composition $R \circ L$ and the identity functor $I_{\mathbf{D}}$, and another between $L \circ R$ and the identity functor $I_{\mathbf{C}}$.

Adjunction is even weaker than equivalence, because it doesn't require that the composition of the two functors be *isomorphic* to the identity functor. Instead it stipulates the existence of a *one way* natural transformation from I_D to $R \circ L$, and another from $L \circ R$ to I_C. Here are the signatures of these two natural transformations:

$$\eta :: I_\mathrm{D} \to R \circ L$$
$$\varepsilon :: L \circ R \to I_\mathrm{C}$$

η is called the unit, and ε the counit of the adjunction.

Notice the asymmetry between these two definitions. In general, we don't have the two remaining mappings:

$$\begin{aligned} R \circ L &\to I_{\mathrm{D}} && \text{not necessarily} \\ I_{\mathrm{C}} &\to L \circ R && \text{not necessarily} \end{aligned}$$

Because of this asymmetry, the functor L is called the *left adjoint* to the functor R, while the functor R is the right adjoint to L. (Of course, left and right make sense only if you draw your diagrams one particular way.)

To summarize what we have done: A categorical product may be defined globally as the *right adjoint* of the diagonal functor:

$$(\mathbf{C} \times \mathbf{C})(\Delta\, c, \langle a, b \rangle) \cong \mathbf{C}(c, a \times b)$$

Java

Meetup