Bartosz Milewski's Programming Cafe

Category Theory, Haskell, Concurrency, C++

March 27, 2019

Promonads, Arrows, and Einstein Notation for Profunctors

Posted by Bartosz Milewski under Category Theory, Haskell, Monads, Programming
1 Comment

i
14 Votes

I've been working with profunctors lately. They are interesting beasts, both in category theory and in programming. In Haskell, they form the basis of profunctor optics–in particular the lens library.

Profunctor Recap

The categorical definition of a profunctor doesn't even begin to describe its richness. You might say that it's just a functor from a product category $\mathbb{C}^{op} \times \mathbb{D}$ to Set (I'll stick to Set for simplicity, but there are generalizations to other categories as well).

A profunctor P (a.k.a., a distributor, or bimodule) maps a pair of objects, c from $\mathbb{C}$ and d from $\mathbb{D}$, to a set $P(c, d)$. Being a functor, it also maps any pair of morphisms in $\mathbb{C}^{op} \times \mathbb{D}$:

$$f \colon c' \to c$$
$$g \colon d \to d'$$

to a function between those sets:

$$P(f, g) \colon P(c, d) \to P(c', d')$$

Notice that the first morphism f goes in the opposite direction to what we normally expect for functors. We say that the profunctor is *contravariant* in its first argument and *covariant* in the second.

But what's so special about this particular combination of source and target categories?

Hom-Profunctor

The key point is to realize that a profunctor generalizes the idea of a hom-functor. Like a profunctor, a hom-functor maps pairs of objects to sets. Indeed, for any two objects in $\mathbb{C}$ we have the set of morphisms between them, $C(a, b)$.

Also, any pair of morphisms in $\mathbb{C}$:

$$f \colon a' \to a \\ g \colon b \to b'$$

can be lifted to a function, which we will denote by $C(f, g)$, between hom-sets:

$$C(f, g) \colon C(a, b) \to C(a', b')$$

Indeed, for any $h \in C(a, b)$ we have:

$$C(f, g) h = g \circ h \circ f \in C(a', b')$$

This (plus functorial laws) completes the definition of a functor from $\mathbb{C}^{op} \times \mathbb{C}$ to Set. So a hom-functor is a special case of an endo-profunctor (where $\mathbb{D}$ is the same as $\mathbb{C}$). It's contravariant in the first argument and covariant in the second.

For Haskell programmers, here's the definition of a profunctor from Edward Kmett's `Data.Profunctor` library:

```
class Profunctor p where
  dimap :: (a' -> a) -> (b -> b') -> p a b -> p a' b'
```

The function `dimap` does the lifting of a pair of morphisms.

Here's the proof that the hom-functor which, in Haskell, is represented by the arrow `->`, is a profunctor:

```
instance Profunctor (->) where
  dimap ab cd bc = cd . bc . ab
```

Not only that: a general profunctor can be considered an extension of a hom-functor that forms a bridge between two categories. Consider a profunctor P spanning two categories $\mathbb{C}$ and $\mathbb{D}$:

$$P \colon \mathbb{C}^{op} \times \mathbb{D} \to Set$$

For any two objects from one of the categories we have a regular hom-set. But if we take one object c from $\mathbb{C}$ and another object d from $\mathbb{D}$, we can generate a set $P(c, d)$. This set works just like a hom-set. Its elements are called *heteromorphisms*, because they can be thought of as representing morphism between two different categories. What makes them similar to morphisms is that they can be composed with regular morphisms. Suppose you have a morphism in $\mathbb{C}$:

$$f \colon c' \to c$$

and a heteromorphism $h \in P(c, d)$. Their composition is another heteromorphism obtained by lifting the pair (f, id_d). Indeed:

$$P(f, id_d) \colon P(c, d) \to P(c', d)$$

so its action on h produces a heteromorphism from c' to d, which we can call the *composition* $h \circ f$ of a heteromorphism h with a morphism f. Similarly, a morphism in $\mathbb{D}$:

$$g \colon d \to d'$$

can be composed with h by lifting (id_c, g).

In Haskell, this new composition would be implemented by applying `dimap f id` to precompose `p c d` with

```
f :: c' -> c
```

and `dimap id g` to postcompose it with

```
g :: d -> d'
```

This is how we can use a profunctor to glue together two categories. Two categories connected by a profunctor form a new category known as their *collage*.

A given profunctor provides unidirectional flow of heteromorphisms from $\mathbb{C}$ to $\mathbb{D}$, so there is no opportunity to compose two heteromorphisms.

Profunctors As Relations

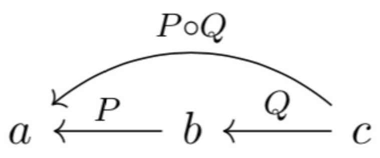
The opportunity to compose heteromorphisms arises when we decide to glue more than two categories. The clue as how to proceed comes from yet another interpretation of profunctors: as proof-relevant relations. In classical logic, a *relation* between sets assigns a Boolean *true* or *false* to each pair of elements. The elements are either related or not, period. In proof-relevant logic, we are not only interested in whether something is true, but also in gathering witnesses to the proofs. So, instead of assigning a single Boolean to each pair of elements, we assign a whole set. If the set is empty, the elements are unrelated. If it's non-empty, each element is a separate witness to the relation.

This definition of a relation can be generalized to any category. In fact there is already a natural relation between objects in a category–the one defined by hom-sets. Two objects a and b are related this way if the hom-set $C(a, b)$ is non-empty. Each morphism in $C(a, b)$ serves as a witness to this relation.

With profunctors, we can define proof-relevant relations between objects that are taken from different categories. Object c in $\mathbb{C}$ is related to object d in $\mathbb{D}$ if $P(c, d)$ is a non-empty set. Moreover, each element of this set serves as a witness for the relation. Because of functoriality of P, this relation is compatible with the categorical structure, that is, it composes nicely with the relation defined by hom-sets.

In general, a composition of two relations P and Q, denoted by $P \circ Q$ is defined as a path between objects. Objects a and c are related if there is a go-between object b such that both $P(a, b)$ and $Q(b, c)$ are non-empty. As a witness of this relation we can pick any pair of elements, one from $P(a, b)$ and one from $Q(b, c)$.

By convention, a profunctor $P(a, b)$ is drawn as an arrow (often crossed) from b to a, $a \nleftarrow b$.


$$\begin{tikzcd} a & b \arrow[l, "P"'] & c \arrow[l, "Q"'] \arrow[ll, bend right, "P\circ Q"'] \end{tikzcd}$$

Composition of profunctors/relations

Profunctor Composition

To create a set of all witnesses of $P \circ Q$ we have to sum over all possible intermediate objects and all pairs of witnesses. Roughly speaking, such a sum (modulo some identifications) is expressed categorically as a coend:

$$(P \circ Q)(a, c) = \int^b P(a, b) \times Q(b, c)$$

As a refresher, a coend of a profunctor P is a set $\int^a P(a, a)$ equipped with a family of injections

$$i_x \colon P(x, x) \to \int^a P(a, a)$$

that is universal in the sense that, for any other set s and a family:

$$\alpha_x \colon P(x, x) \to s$$

there is a unique function h that factorizes them all:

$$\alpha_x = h \circ i_x$$

$$\begin{tikzcd}
& P(x,x) \arrow[dl, "i_x"'] \arrow[ddl, "\alpha_x"] \\
\int^a P(a,a) \arrow[d, dashed, "h"] & \\
s &
\end{tikzcd}$$

Universal property of a coend

Profunctor composition can be translated into pseudo-Haskell as:

```
type Procompose q p a c = exists b. (p a b, q b c)
```

where the coend is encoded as an existential data type. The actual implementation (again, see Edward Kmett's `Data.Profunctor.Composition`) is:

```
data Procompose q p a c where
  Procompose :: q b c -> p a b -> Procompose q p a c
```

The existential quantifier is expressed in terms of a GADT (Generalized Algebraic Data Type), with the free occurrence of `b` inside the data constructor.

Einstein's Convention

By now you might be getting lost juggling the variances of objects appearing in those formulas. The coend variable, for instance, must appear under the integral sign once in the covariant and once in the contravariant position, and the variances on the right must match the variances on the left. Fortunately, there is a precedent in a different branch of mathematics, tensor calculus in vector spaces, with the kind of notation that takes care of variances. Einstein coopted and expanded this notation in his theory of relativity. Let's see if we can adapt this technique to the calculus of profunctors.

The trick is to write contravariant indices as superscripts and the covariant ones as subscripts. So, from now on, we'll write the components of a profunctor p (we'll switch to lower case to be compatible with Haskell) as $p^c{}_d$. Einstein also came up with a clever convention: implicit summation over a repeated index. In the case of profunctors, the summation corresponds to taking a coend. In this notation, a coend over a profunctor p looks like a trace of a tensor:

$$p^a{}_a = \int^a p(a,a)$$

The composition of two profunctors becomes:

$$(p \circ q)^a{}_c = p^a{}_b \, q^b{}_c = \int^b p(a,b) \times q(b,c)$$

The summation convention applies only to adjacent indices. When they are separated by an explicit product sign (or any other operator), the coend is not assumed, as in:

$$p^a{}_b \times q^b{}_c$$

(no summation).

The hom-functor in a category $\mathbb{C}$ is also a profunctor, so it can be notated appropriately:

$$C^a{}_b = C(a, b)$$

The co-Yoneda lemma (see Ninja Yoneda) becomes:

$$C^c{}_{c'}\, p^{c'}{}_d \cong p^c{}_d \cong p^c{}_{d'}\, D^{d'}{}_d$$

suggesting that the hom-functors $C^c{}_{c'}$ and $D^{d'}{}_d$ behave like Kronecker deltas (in tensor-speak) or unit matrices. Here, the profunctor p spans two categories

$$p \colon \mathbb{C}^{op} \times \mathbb{D} \to Set$$

The lifting of morphisms:

$$f \colon c' \to c$$
$$g \colon d \to d'$$

can be written as:

$$p^f{}_g \colon p^c{}_d \to p^{c'}{}_{d'}$$

There is one more useful identity that deals with mapping out from a coend. It's the consequence of the fact that the hom-functor is continuous. It means that it maps (co-) limits to limits. More precisely, since the hom-functor is contravariant in the first variable, when we fix the target object, it maps colimits in the first variable to limits. (It also maps limits to limits in the second variable). Since a coend is a colimit, and an end is a limit, continuity leads to the following identity:

$$Set(\int^c p(c, c), s) \cong \int_c Set(p(c, c), s)$$

for any set s. Programmers know this identity as a generalization of case analysis: a function from a sum type is a product of functions (one function per case). If we interpret the coend as an existential quantifier, the end is equivalent to a universal quantifier.

Let's apply this identity to the mapping out from a composition of two profunctors:

$$p^a{}_b\, q^b{}_c \to s = Set\big(\int^b p(a, b) \times q(b, c), s\big)$$

This is isomorphic to:

$$\int_b Set\Big(p(a, b) \times q(b, c), s\Big)$$

or, after currying (using the product/exponential adjunction),

$$\int_b Set\Big(p(a, b), q(b, c) \to s\Big)$$

This gives us the mapping out formula:

$$p^a{}_b q^b{}_c \to s \cong p^a{}_b \to q^b{}_c \to s$$

with the right hand side natural in b. Again, we don't perform implicit summation on the right, where the repeated indices are separated by an arrow. There, the repeated index b is universally quantified (through the end), giving rise to a natural transformation.

Bicategory Prof

Since profunctors can be composed using the coend formula, it's natural to ask if there is a category in which they work as morphisms. The only problem is that profunctor composition satisfies the associativity and unit laws (see the co-Yoneda lemma above) only up to isomorphism. Not to worry, there is a name for that: a *bicategory*. In a bicategory we have objects, which are called 0-cells; morphisms, which are called 1-cells; and morphisms between morphisms, which are called 2-cells. When we say that categorical laws are satisfied up to isomorphism, it means that there is an invertible 2-cell that maps one side of the law to another.

The bicategory $Prof$ has categories as 0-cells, profunctors as 1-cells, and natural transformations as 2-cells. A natural transformation α between profunctors p and q

$$\alpha \colon p \Rightarrow q$$

has components that are functions:

$$\alpha^c{}_d \colon p^c{}_d \to q^c{}_d$$

satisfying the usual naturality conditions. Natural transformations between profunctors can be composed as functions (this is called vertical composition). In fact 2-cells in any bicategory are composable, and there always is a unit 2-cell. It follows that 1-cells between any two 0-cells form a category called the hom-category.

But there is another way of composing 2-cells that's called horizontal composition. In $Prof$, this horizontal composition is not the usual horizontal composition of natural transformations, because composition of profunctors is not the usual composition of functors. We have to construct a natural transformation between one composition of profuntors, say $p^a{}_b q^b{}_c$ and another, $r^a{}_b s^b{}_c$, having at our disposal two natural transformations:

$$\alpha \colon p \Rightarrow r$$

$$\beta \colon q \Rightarrow s$$

The construction is a little technical, so I'm moving it to the appendix. We will denote such horizontal composition as:

$$(\alpha \circ \beta)^a{}_c \colon p^a{}_b q^b{}_c \to r^a{}_b s^b{}_c$$

If one of the natural transformations is an identity natural transformation, say, from $p^a{}_b$ to $p^a{}_b$, horizontal composition is called *whiskering* and can be written as:

$$(p \circ \beta)^a{}_c \colon p^a{}_b \, q^b{}_c \to p^a{}_b \, s^b{}_c$$

Promonads

The fact that a monad is a monoid in the category of endofunctors is a lucky accident. That's because, in general, a monad can be defined in any bicategory, and Cat just happens to be a (strict) bicategory. It has (small) categories as 0-cells, functors as 1-cells, and natural transformations as 2-cells. A monad is defined as a combination of a 0-cell (you need a category to define a monad), an endo-1-cell (that would be an endofunctor in that category), and two 2-cells. These 2-cells are variably called multiplication and unit, μ and η, or `join` and `return`.

Since $Prof$ is a bicategory, we can define a monad in it, and call it a promonad. A promonad consists of a 0-cell C, which is a category; an endo-1-cell p, which is a profunctor in that category; and two 2-cells, which are natural transformations:

$$\mu^a{}_b \colon p^a{}_c \, p^c{}_b \to p^a{}_b$$

$$\eta^a{}_b \colon C^a{}_b \to p^a{}_b$$

Remember that $C^a{}_b$ is the hom-profunctor in the category C which, due to co-Yoneda, happens to be the unit of profunctor composition.

Programmers might recognize elements of the Haskell `Arrow` in it (see my blog post on monoids).

We can apply the mapping-out identity to the definition of multiplication and get:

$$\mu^a{}_b \colon p^a{}_c \to p^c{}_b \to p^a{}_b$$

Notice that this looks very much like composition of heteromorphisms. Moreover, the monadic unit η maps regular morphisms to heteromorphisms. We can then construct a new category, whose objects are the same as the objects of $\mathbb{C}$, with hom-sets given by the profunctor p. That is, a hom set from a to b is the set $p^a{}_b$. We can define an identity-on-object functor J from $\mathbb{C}$ to that category, whose action on hom-sets is given by η.

Interestingly, this construction also works in the opposite direction (as was brought to my attention by Alex Campbell). Any indentity-on-objects functor defines a promonad. Indeed, given a functor J, we can always turn it into a profunctor:

$$p(c, d) = D(J\,c, J\,d)$$

In Einstein notation, this reads:

$$p^c{}_d = D^{J\,c}{}_{J\,d}$$

Since J is identity on objects, the composition of morphisms in D can be used to define the composition of heteromorphisms. This, in turn, can be used to define μ, thus showing that p is a promonad on $\mathbb{C}$.

Conclusion

I realize that I have touched upon some pretty advanced topics in category theory, like bicategories and promonads, so it's a little surprising that these concepts can be illustrated in Haskell, some of them being present in popular libraries, like the `Arrow` library, which has applications in functional reactive programming.

I've been experimenting with applying Einstein's summation convention to profunctors, admittedly with mixed results. This is definitely work in progress and I welcome suggestions to improve it. The main problem is that we sometimes need to apply the sum (coend), and at other times the product (end) to repeated indices. This is in particular awkward in the formulation of the mapping out property. I suggest separating the non-summed indices with product signs or arrows but I'm not sure how well this will work.

Appendix: Horizontal Composition in Prof

We have at our disposal two natural transformations:

$$\alpha \colon p \Rightarrow r$$

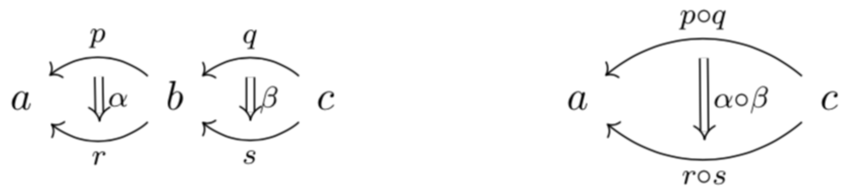
$$\beta \colon q \Rightarrow s$$

and the following coend, which is the composition of the profunctors p and q :

$$\int^b p(a, b) \times q(b, c)$$

Our goal is to construct an element of the target coend:

$$\int^b r(a, b) \times s(b, c)$$



$$\begin{tikzcd} a & b \arrow[l, bend right, "p"', ""{name=P, below}] \arrow[l, bend left, "r", ""{name=R, above}] & c \arrow[l, bend right, "q"', ""{name=Q, below}] \arrow[l, bend left, "s", ""{name=S, above}] \arrow[Rightarrow, from=P, to=R, "\alpha"] \arrow[Rightarrow, from=Q, to=S, "\beta"] \end{tikzcd} \qquad \begin{tikzcd} a & & c \arrow[ll, bend right, "p\circ q"', ""{name=U, below}] \arrow[ll, bend left, "r\circ s", ""{name=D, above}] \arrow[Rightarrow, from=U, to=D, "\alpha\circ\beta"] \end{tikzcd}$$

Horizontal composition of 2-cells

To construct an element of a coend, we need to provide just one element of $r(a, b') \times s(b', c)$ for some b' . We'll look for a function that would construct such an element in the following hom-set:

$$Set\Big(\int^b p(a, b) \times q(b, c), r(a, b') \times s(b', c)\Big)$$

Using Einstein notation, we can write it as:

$$p^a{}_b\, q^b{}_c \to r^a{}_{b'} \times s^{b'}{}_c$$

and then use the mapping out property:

$$p^a{}_b \to q^b{}_c \to r^a{}_{b'} \times s^{b'}{}_c$$

We can pick b' equal to b and implement the function using the components of the two natural transformations, $\alpha^a{}_b \times \beta^b{}_c$.

Of course, this is how a programmer might think of it. A mathematician will use the universal property of the coend $(p \circ q)^a{}_c$, as in the diagram below (courtesy Alex Campbell).

$$\begin{tikzcd}
& p^a{}_b \times q^b{}_c \arrow[dl, "i_b"'] \arrow[ddl, "in_b \circ (\alpha \times \beta)"] \\
(p \circ q)^a{}_c \arrow[d, dashed, "(\alpha \circ \beta)^a{}_c"'] & \\
(r \circ s)^a{}_c &
\end{tikzcd}$$

Horizontal composition using the universal property of a coend

In Haskell, we can define a natural transformation between two (endo-) profunctors as a polymorphic function:

```
newtype PNat p q = PNat (forall a b. p a b -> q a b)
```

Horizontal composition is then given by:

```
horPNat :: PNat p r -> PNat q s -> Procompose p q a c
        -> Procompose r s a c
horPNat (PNat alpha) (PNat beta) (Procompose pbc qdb) =
  Procompose (alpha pbc) (beta qdb)
```

Acknowledgment

I'm grateful to Alex Campbell from Macquarie University in Sydney for extensive help with this blog post.

Further Reading

- Dominique Bourn et Jacques Penon, 2-Catégories Réductibles
- Ross Street, Cauchy characterization of enriched categories

One Response to “Promonads, Arrows, and Einstein Notation for Profunctors”

1. (Milewski) Промонады, стрелки и нотация Эйнштейна для профункторов — Переводной календарь Says:

   November 14, 2019 at 1:17 am

   […] Перевод статьи Бартоша Милевски «Promonads, Arrows, and Einstein Notation for Profunctors» (исходный текст расположен по адресу — Текст оригинальной статьи). […]