FILTERED COLIMITS

BARTOSZ MILEWSKI

Previously we were exploring universal constructions for products, coproducts, and exponentials. In particular, we were able to prove the distributive law:

$$(a + b) \times c \cong a \times c + b \times c$$

The power of this law is that it relates the mapping-in universal construction (product on the left) with the mapping-out one (coproduct on the right). If you take into account that products and coproducts are just special cases of limits and colimits, you may ask a more general question: under what conditions limits commute with colimits. In a cartesian closed category a product of sums is not equal to the sum of products:

$$(a + b) \times (c + d) \ncong a \times c + b \times d$$

So, in general, products don't commute with coproducts. But if you replace coproducts with a special kind of colimits, then it can be shown that:

Theorem 1. *In Set, filtered colimits commute with finite limits.*

In this post I'll try to explain these terms and provide some intuition why it works and how filtered colimits are related to the more traditional notion of limits that we know from calculus.

1. Limits

Let's start with limits. They are like products, except that, instead of just two objects at the bottom, you have any number of objects plus a bunch of morphisms between them. That's called a diagram. Then you have an apex with arrows going down to all the objects in the diagram; and you get what is called a cone. If you have morphisms in your diagram, they form triangles. These triangles must commute. For instance, in Fig 1, we have:

$$g \circ \pi_1 = \pi_3$$

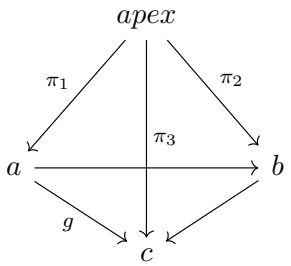

Figure 1. A cone

This means that not all projections are independent–that you may obtain one projection from another by post-composing it with a morphism from the diagram. In Fig 1, for instance, you may extract a value of c either directly using π_3 or by applying g to the result of π_1.

A limit is defined as the universal cone with the apex Lim. It means that, if you have any other cone with some apex c, built over the same diagram, there is a unique morphism h from c to Lim that makes all the triangles commute. For instance, in Fig 2, one of the commuting conditions is:

$$\pi_1 \circ h = f_1$$

and so on. We've seen similar commuting conditions in the definition of the product.

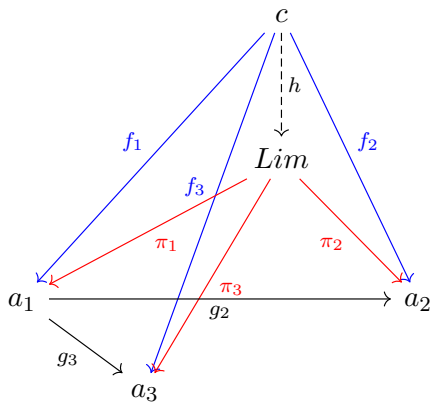

FIGURE 2. A universal cone

If you think of Lim in this example as a data structure, you would implement it as a product of a_1, a_2, and a_3, together with two functions:

$$g_3 : a_1 \to a_3$$

$$g_2 : a_1 \to a_2$$

But because of the commuting conditions, the three values stored in Lim cannot be independent. If you pick a value for a_1, then the values for a_2 and a_3 are uniquely determined.

A limit, just like a product, is defined by a mapping-in property. If you want to define a morphism from some c to Lim, you need to provide three morphisms f_1, f_2, and f_3. However, unlike in the case of a product, these morphisms must satisfy some commuting conditions. Here, f_3 must be equal to $g_3 \circ f_1$ and $f_2 = g_2 \circ f_1$. So, really, you only need to define f_1, and that uniquely determines h. This is why the cones in Fig 2 can be simplified, as shown in Fig 3.

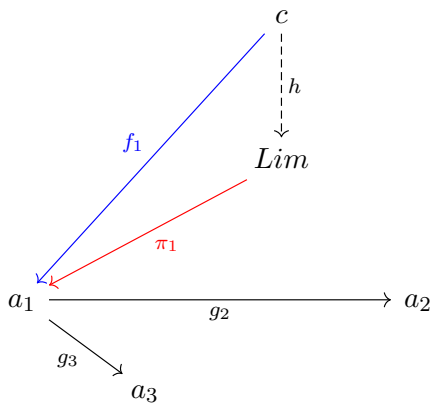

FIGURE 3. A simplified universal cone

Notice that the diagram essentially forms a subcategory inside the category C, even if we don't explicitly draw all the identity morphisms or all the compositions. This is because triangles built by composing commuting triangles are again commuting. It therefore makes sense to define a diagram as a functor F from an (often much smaller) index category J to C. In our case it would be a category with just three objects, j_1, j_2, j_3, and two non-identity morphisms. (The diagram category for the product is even simpler: just two objects, no non-trivial morphisms.)

The properties of the diagram category determine the nature of cones and the nature of the limits. For instance, functors from a finite category will produce *finite limits*.

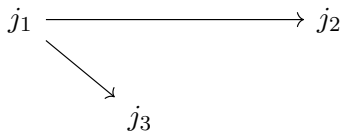

FIGURE 4. Diagram category J

The diagram category J in our example has a very peculiar property: it has a cone for every pair of objects (it's a cone inside J, not to be confused with the cone in C). For instance, the pair j_2, j_3 is part of the cone with the apex j_1. This is also the apex for the (somewhat degenerate) cone based on j_1 and j_2 (with or without the connecting morphism). A category in which there is a cone for every finite subdiagram is called *cofiltered*. Limits defined by functors from cofiltered categories are called *cofiltered limits*.

The intuition is that cofiltered categories exhibit some kind of ordering. You may think of j_1 as a lower bound of j_2 and j_3. Following these bounds, you might eventually get to some kind of roots–here it's the object j_1–and these roots will dictate the behavior of cones and the behavior of limits. Things get really interesting when the diagram category is infinite, because then there is no guarantee that you'll ever reach a root. There is, for instance, no smallest (negative) integer, even though integers are ordered. You can begin to see parallels with traditional limits, like:

$$\lim_{j \to -\infty} a_j$$

That's where these ideas originally came from.

Limits in the category of sets have a particularly simple interpretation. In *Set*, we can use functions from the terminal object–the singleton set–to pick individual set elements.

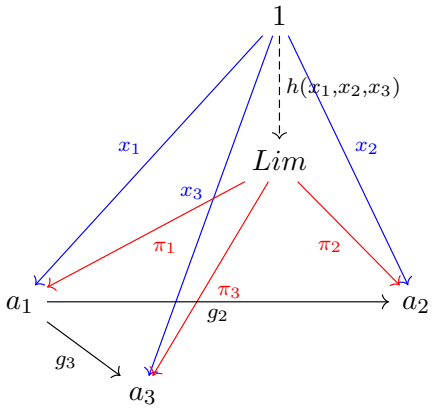

FIGURE 5. Elements of the limit

For every selection in Fig 5. of x_1, x_2, x_3 there is a unique $h(x_1, x_2, x_3)$ that picks an element in *Lim*. But a selection of x_1, x_2, x_3 is nothing but a cone with the apex 1. So there is a one-to-one correspondence between elements of *Lim* and such cones. In other words, *Lim* is a *set of apex-1 cones*.

2. Colimits

Colimits are dual to limits–you get them by inverting all the arrows. So, instead of projections, you get injections, and the universal condition defines a *mapping out* of a colimit (see Fig 6).

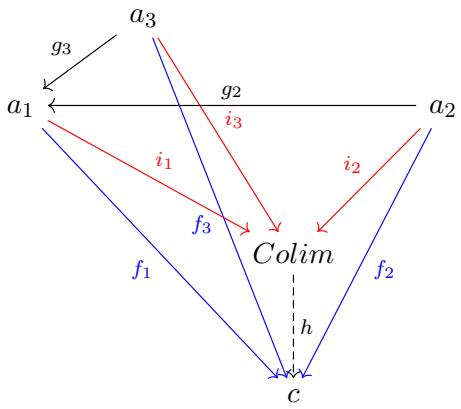

Figure 6. A universal cocone

If you look at the colimit as a data structure, it is similar to a coproduct, except that not all the injections are independent. In the example in Fig 6, i_3 and i_2 are determined by pre-composing i_1 with g_3 and g_2, respectively. It's not clear how to implement a colimit in Haskell, so here's a pseudo-Haskell attempt using imaginary dependent-type syntax:

```
data Colim a1 a2 a3 (g2 :: a2 -> a1) (g3 :: a3 -> a1) =
         = A1 a1 | A2 a2 | A3 a3
```

To deconstruct this colimit, you only need to provide one function $f_1 : a_1 \to c$.

```
h :: (a1 -> c) -> Colim a1 a2 a3 g2 g3 -> c
h f1 (A1 a1)    = f1 a1
h f1 (A2 a2) = f1 (g2 a2)
h f1 (A3 a3) = f1 (g3 a3)
```

Granted, in a lazy language like Haskell, this would be an overkill way to store essentially just one value.

A colimit in the category of sets simplifies to a disjoint union of sets, in which some elements are identified. Suppose that the colimit $Colim_J F$ is defined by some diagram category J and a functor $F : J \to Set$. Each object j in J produces a set Fj.

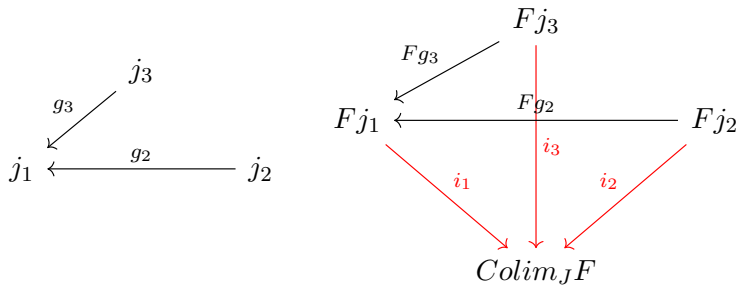

FIGURE 7. Colimit in Set. On the left, the diagram category J.

The disjoint union of all these sets is a set whose elements are the pairs (x, j) where $x \in Fj$. (Notice that the sets may overlap, but each element from the overlap will be counted as many times as the number of sets it belongs to.) Coproduct injections are then functions that take an element $x \in Fj$ and map it into an element $(x, j) \in Colim_J F$. But that doesn't take into account the presence of morphisms in the diagram. These morphisms are mapped to functions between corresponding sets. For instance, in Fig 7, we can take an element $x \in Fj_2$. It is injected, using i_2, as an element $(x, j_2) \in Colim_J F$. But there is another path from Fj_2 that uses Fg_2 followed by i_1. That produces $((Fg_2)x, j_1)$. If the triangle is to commute, these two must be equal. So in the actual colimit, they must be identified. In general, any two elements of the disjoint union that satisfy this relation:

$$(x, j) \rightsquigarrow (x', j') \text{ if } \exists_{g:j\to j'}(Fg)x = x'$$

must be identified. This is not an equivalence relation, but it can be extended to one (by first symmetrizing it, and then making it transitive again). A colimit is then a quotient of the disjoint union by this equivalence.

As before, I chose this example to illustrate a special type of a diagram. This is a diagram that can be obtained using a functor from a *filtered* category. A filtered category has this property that for any finite subdiagram, there is a cocone under it. Here, for instance, the subdiagram formed by j_2 and j_3 has a cocone with the apex j_1. Again, you may think of j_1 as a kind of upper bound of j_2 and j_3. If the filtered category is finite, following upper bounds will eventually lead you to some roots. And in Set, the equivalence relation will allow you shift all the elements down to those roots. But in an infinite case (think natural numbers) there may be no largest element–no root. And that brings filtered colimits closer to the intuition we have for limits in calculus. In fact, all the interesting filtered colimits are based on infinite diagrams.

3. Commuting Limits and Colimits

What does it mean for a limit to commute with a colimit? A single colimit is generated by a functor from some index category $I \to C$. What we need is a bunch of such colimits so that we can take a limit over those. Therefore we need a bunch of functors $I \to C$. Moreover, those colimits have to form a diagram. So we need another index category J to parameterize those functors. Altogether, we need a functor of two arguments:

$$F : I \times J \to C$$

It follows that, for any given j in J we have a functor $F(-, j) : I \to C$. We can take a colimit of that. Then we gather those colimits into a diagram whose shape is defined by J, and then take its limit. We get:

$$Lim_J(Colim_I F)$$

Alternatively, when we fix some i in I, we get a functor $F(i,-) : J \to C$. We can take a limit of that. Then we can gather all those limits and form a diagram whose shape is defined by I. Finally we can take a colimit of that:

$$Colim_I(Lim_J F)$$

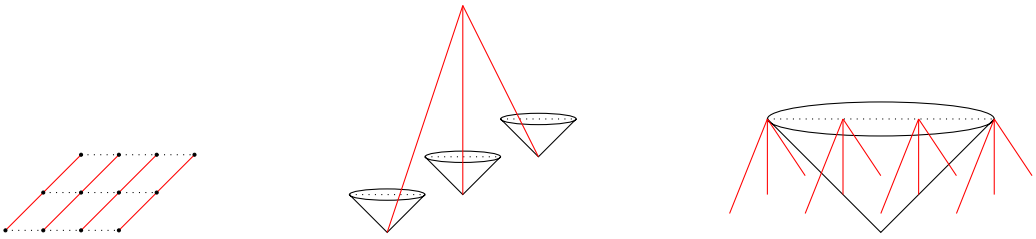

FIGURE 8. Commuting limits (red diagram of shape J) and colimits (black diagram of shape I)

It's not difficult to construct the mapping:

$$Colim_I(Lim_J F) \to Lim_J(Colim_I F)$$

using the universal property, since the colimit has the mapping-out property. It's the other way around that's tricky. But it always works in the special case when I is filtered, J is finite, and C is *Set*.

Here's the sketch of this amazing proof, which you can find in Saunders Mac Lane's *Categories for the Working Mathematician*.

Since the target of the functor is Set, it might help to visualize its image as a rectangular array of sets. A fixed j picks up a row of such sets, whereas a fixed i picks up a column. Because we are dealing with sets, we can try to define the mapping:

$$Lim_J(Colim_I F) \to Colim_I(Lim_J F)$$

pointwise. Let's pick an element of the limit on the left. As we've established earlier, a limit in Set is a set of apex-1 cones. So let's pick one such cone. It's just a selection of elements from a bunch of colimits.

As we've seen before, a colimit in Set is a discriminated union with some identifications. So our apex-1 cone will pick a set of representatives, one per colimit, say (x_n, i_n). Any time there is a morphism $g : i_n \to i'_n$, we can replace one representative with another $(g(x_n), i'_n)$. The intuition is that we can slide the representatives horizontally within each row along morphisms.

If I is a filtered category, then for any finite number of objects i_n, we can always find a common root (it will be the apex i of a cocone formed by i_n in I). So we can slide all the representatives to a single column. In other words, our cone can be brought to a set of representatives (y_n, i), with a common i.

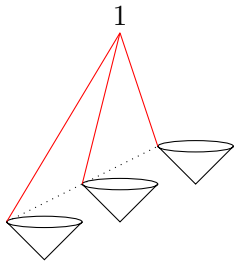

FIGURE 9. A single cone after shifting representatives from all colimits to a common column

But that's just a cone over J. It's an element of $Lim_J F$. And we can inject it into a colimit over I to get an element of $Colim_I(Lim_J F)$. We have thus defined our mapping.

4. Conclusion

If you didn't get the proof the first time, don't get discouraged. Take a break, sleep over it, and then read it slowly again. Make sure you have internalized all the definitions. Draw your own picyures. The two major tricks are: (1) visualizing an element of a limit as a cone originating from the singleton set, and (2) the idea of sliding the elements of multiple colimits to a common column.

The importance of this theorem is that it tells you when and how you can define mappings out of limits. For instance, how to define functions from a product or from an end.

5. Acknowledgment

I'm grateful to Derek Elkins for correcting mistakes in the original version of this post.