Chapter 5

Categories at work

We have now set up an understanding of the basic notions of category theory: categories, functors, natural transformations, and universal properties. We have discussed many sources of examples: orders, graphs, monoids, and databases. We begin this chapter with the notion of *adjoint functors* (also known as *adjunctions*), which are like dictionaries that translate back and forth between different categories.

5.1 Adjoint functors

Just above, in the introduction to this chapter, I said that adjoint functors are like dictionaries that translate back and forth between different categories. How far can we take that analogy?

In the common understanding of dictionaries, we assume that the two languages (say French and English) are equally expressive, and that a good dictionary will be an even exchange of ideas. But in category theory we often have two categories that are not on the same conceptual level. This is most clear in the case of so-called *free-forgetful adjunctions*. In Section 5.1.1 we will explore the sense in which each adjunction provides a dictionary between two categories that are not necessarily on an equal footing, so to speak.

5.1.1 Discussion and definition

Consider the category of monoids and the category of sets. A monoid $(M, e, \star)$ is a set with an identity element and a multiplication formula that is associative. A set is just a set. A dictionary between **Mon** and **Set** should not be required to set up an even exchange, but instead an exchange that is appropriate to the structures at hand. It will be in the form of two functors, one we'll denote by $L\colon \mathbf{Set} \to \mathbf{Mon}$, and one we'll denote by $R\colon \mathbf{Mon} \to \mathbf{Set}$. But to say what "appropriate" means requires more work.

Let's bring it down to earth with an analogy. A one-year-old can make repeatable noises and an adult can make repeatable noises. One might say "after all, talking is nothing but making repeatable noises." But the adult's repeatable noises are called words, they form sentences, and these sentences can cause nuclear wars. There is something more in adult language than there is simply in repeatable sounds. In the same vein, a tennis match can be viewed as physics, but you won't see the match. So we have something analogous to two categories here: ((repeated noises)) and ((meaningful words)).

We are looking for adjoint functors going back and forth, serving as the appropriate sort of dictionary.

To translate baby talk into adult language we would make every repeated noise a kind of word, thereby granting it meaning. We don't know what a given repeated noise should mean, but we give it a slot in our conceptual space, always pondering "I wonder what she means by Konnen.." On the other hand, to translate from meaningful words to repeatable noises is easy. We just hear the word as a repeated noise, which is how the baby probably hears it.

Adjoint functors often come in the form of "free" and "forgetful". Here we freely add Konnen to our conceptual space without having any idea how it adheres to the rest of the child's noises or feelings. But it doesn't act like a sound to us, it acts like a word; we don't know what it means but we figure it means something. Conversely, the translation going the other way is "forgetful", forgetting the meaning of our words and just hearing them as sounds. The baby hears our words and accepts them as mere sounds, not knowing that there is anything extra to get.

Back to sets and monoids, the sets are like the babies from our story: they are simple objects full of unconnected dots. The monoids are like adults, forming words and performing actions. In the monoid, each element means something and combines with other elements in some way. There are lots of different sets and lots of different monoids, just as there are many babies and many adults, but there are patterns to the behavior of each kind and we put them in different categories.

Applying free functor $L\colon \mathbf{Set} \to \mathbf{Mon}$ to a set X makes every element $x \in X$ a word, and these words can be strung together to form more complex words. (We discussed the free functor in Section 3.1.1.12.) Since a set such as X carries no information about the meaning or structure of its various elements, the free monoid $F(X)$ does not relate different words in any way. To apply the forgetful functor $R\colon \mathbf{Mon} \to \mathbf{Set}$ to a monoid, even a structured one, is to simply forget that its elements are anything but mere elements of a set. It sends a monoid $(M, 1, \star)$ to the set M.

The analogy is complete. However, this is all just ideas. Let's give a definition, then return to our sets, monoids, sounds, and words.

Definition 5.1.1.1. Let $\mathcal{B}$ and $\mathcal{A}$ be categories. [1] An *adjunction between $\mathcal{B}$ and $\mathcal{A}$* is a pair of functors

$$L\colon \mathcal{B} \to \mathcal{A} \qquad \text{and} \qquad R\colon \mathcal{A} \to \mathcal{B}$$

together with a natural isomorphism [2] whose component for any objects $A \in \mathrm{Ob}(\mathcal{A})$ and $B \in \mathrm{Ob}(\mathcal{B})$ is:

$$\alpha_{B,A}\colon \mathrm{Hom}_{\mathcal{A}}(L(B), A) \xrightarrow{\cong} \mathrm{Hom}_{\mathcal{B}}(B, R(A)). \tag{5.1}$$

This isomorphism is called the *adjunction isomorphism* for the (L, R) adjunction, and for any morphism $f\colon L(B) \to A$ in $\mathcal{A}$, we refer to $\alpha_{B,A}(f)\colon B \to R(A)$ as *the adjunct* of f. [3]

[1] Throughout this definition, notice that B's come before A's, especially in (5.1), which might be confusing. It was a stylistic choice to match with the **B**abies and **A**dults discussion above and below this definition.

[2] The natural isomorphism α (see Lemma 4.3.2.12) is between two functors $\mathcal{B}^{\mathrm{op}} \times \mathcal{A} \to \mathbf{Set}$, namely the functor $(B, A) \mapsto \mathrm{Hom}_{\mathcal{A}}(L(B), A)$ and the functor $(B, A) \mapsto \mathrm{Hom}_{\mathcal{B}}(B, R(A))$.

[3] Conversely, for any $g\colon B \to R(A)$ in $\mathcal{B}$ we refer to $\alpha^{-1}_{B,A}(g)\colon L(B) \to A$ as *the adjunct* of g.

The functor L is called the *left adjoint* and the functor R is called the *right adjoint*. We may say that L *is the left adjoint of* R or that R *is the right adjoint of* L. [4] We often denote this setup by

$$L\colon \mathcal{B} \rightleftarrows \mathcal{A} : R$$

Proposition 5.1.1.2. *Let* $L\colon \mathbf{Set} \to \mathbf{Mon}$ *be the functor sending* $X \in \mathrm{Ob}(\mathbf{Set})$ *to the free monoid* $L(X) := (\mathrm{List}(X), [\,], +\!+)$*, as in Definition 3.1.1.15. Let* $R\colon \mathbf{Mon} \to \mathbf{Set}$ *be the functor sending each monoid* $\mathcal{M} := (M, 1, \star)$ *to its underlying set* $R(\mathcal{M}) := M$*. Then* L *is left adjoint to* R*.*

Proof. If we can find a natural isomorphism of sets

$$\alpha_{X,\mathcal{M}}\colon \mathrm{Hom}_{\mathbf{Mon}}(L(X), \mathcal{M}) \to \mathrm{Hom}_{\mathbf{Set}}(X, R(\mathcal{M}))$$

we will have succeeded in showing that these functors are adjoint.

Suppose given an element $f \in \mathrm{Hom}_{\mathbf{Mon}}(L(X), \mathcal{M})$, i.e. a monoid homomorphism $f\colon \mathrm{List}(X) \to M$ (sending $[\,]$ to 1 and list concatenation to $\star$). Then in particular we can apply f to the singleton list $[x]$ for any $x \in X$. This gives a function $X \to M$ by $x \mapsto f([x])$, and this is $\alpha_{X,\mathcal{M}}(f)\colon X \to M = R(\mathcal{M})$. We need only to supply an inverse $\beta_{X,\mathcal{M}}\colon \mathrm{Hom}_{\mathbf{Set}}(X, R(\mathcal{M})) \to \mathrm{Hom}_{\mathbf{Mon}}(L(X), \mathcal{M})$.

Suppose given an element $g \in \mathrm{Hom}_{\mathbf{Set}}(X, R(\mathcal{M}))$, i.e. a function $g\colon X \to M$. Then to any list $\ell = [x_1, x_2, \ldots, x_n] \in \mathrm{List}(X)$ we can assign $\beta_{X,\mathcal{M}}(\ell) := g(x_1) \star g(x_2) \star \cdots \star g(x_n)$ (if $\ell = [\,]$ is the empty list, assign $\beta_{X,\mathcal{M}}([\,]) := 1$). We now have a function $\mathrm{List}(X) \to M$. It is a monoid homomorphism because it respects identity and composition. It is easy to check that β and α are mutually inverse, completing the proof.

□

Example 5.1.1.3. We need to ground our discussion in some concrete mathematics. In Proposition 5.1.1.2 we provided our long-awaited adjunction between sets and monoids. A set X gets transformed into a monoid by considering lists in X; a monoid $\mathcal{M}$ gets transformed into a set by forgetting the multiplication law. So we have a functor going one way and the other,

$$L\colon \mathbf{Set} \to \mathbf{Mon}, \qquad\qquad R\colon \mathbf{Mon} \to \mathbf{Set},$$

but an adjunction is more than that: it includes a guarantee about the relationship between these two functors. What is the relationship between L and R? Consider an arbitrary monoid $\mathcal{M} = (M, 1, *)$.

If I want to pick out 3 elements of the set M, that's the same thing as giving a function $\{a, b, c\} \to M$. But that function exists in the category of sets; in fact it is an element of $\mathrm{Hom}_{\mathbf{Set}}(\{a, b, c\}, M)$. But since $M = R(\mathcal{M})$ is the underlying set of our monoid, we can view the current paragraph in the light of our adjunction Equation (5.1) by saying it has been about the set

$$\mathrm{Hom}_{\mathbf{Set}}(\{a, b, c\}, R(\mathcal{M})).$$

This set classifies all the ways to pick three elements out of the underlying set of our monoid $\mathcal{M}$. It was constructed completely from within the category **Set**.

[4] The left adjoint does not have to be called L, nor does the right adjoint have to be called R, of course. This is suggestive.

Now we ask what Equation (5.1) means. The equation

$$\mathrm{Hom}_{\mathbf{Mon}}(L(\{a,b,c\}),\mathcal{M}) \cong \mathrm{Hom}_{\mathbf{Set}}(\{a,b,c\},R(\mathcal{M})).$$

tells us that somehow we can answer the same question completely from within the category of monoids. In fact it tells us how to do so, namely as $\mathrm{Hom}_{\mathbf{Mon}}(\mathrm{List}(\{1,2,3\},\mathcal{M})$. Exercise 5.1.1.4 looks at how that should go. The answer is "hidden" in the proof of Proposition 5.1.1.2.

Exercise 5.1.1.4. Let $X = \{a,b,c\}$ and let $\mathcal{M} = (\mathbb{N},1,*)$ be the multiplicative monoid of natural numbers (see Example 3.1.3.2). Let $f\colon X \to \mathbb{N}$ be the function given by $f(a) = 7, f(b) = 2, f(c) = 2$, and let $\beta_{X,\mathcal{M}}\colon \mathrm{Hom}_{\mathbf{Set}}(X,R(\mathcal{M})) \to \mathrm{Hom}_{\mathbf{Mon}}(L(X),\mathcal{M})$ be as in the proof of Proposition 5.1.1.2. What is $\beta_{X,\mathcal{M}}(f)([b,b,a,c])$? ◊

Let us look once more at the adjunction between adults and babies. Using the notation of Definition 5.1.1.1 $\mathcal{A}$ is the "adult category" of meaningful words and $\mathcal{B}$ is the "baby category" of repeated noises. The left adjoint turns every repeated sound into a meaningful word (having "free" meaning) and the right adjoint "forgets" the meaning of any word and considers it merely as a sound.

At the risk of taking this simple analogy too far, let's have a go at the heart of the issue: how to conceive of the isomorphism (5.1) of Hom's. Once we have freely given a slot to each of baby's repeated sounds, we try to find a mapping from the lexicon $L(B)$ of these new words to our own lexicon A of meaningful words; these are mappings in the adult category $\mathcal{A}$ of the form $L(B) \to A$. And (stretching it) the baby tries to find a mapping (which we might see as emulation) from her set B of repeatable sounds to the set $R(A)$ of the sounds the adult seems to repeat. If there was a global system for making these transformations that would establish (5.1) and hence the adjunction.

Note that the directionality of the adjunction makes a difference. If $L\colon \mathcal{B} \to \mathcal{A}$ is left adjoint to $R\colon \mathcal{A} \to \mathcal{B}$ we rarely have an isomorphism $\mathrm{Hom}_{\mathcal{A}}(A,L(B)) \cong \mathrm{Hom}_{\mathcal{B}}(R(A),B)$. In the case of babies and adults, we see that it would make little sense to look for a mapping in the category of meaningful words from the adult lexicon to the wordifications of baby-sounds, because there is unlikely to be a good candidate for most of our words. That is, to which of our child's repeated noises would we assign the concept "weekday"?

Again, the above is simply an analogy, and almost certainly not formalizable. The next example shows mathematically the point we tried to make in the previous paragraph, that the directionality of an adjunction is not arbitrary.

Example 5.1.1.5. Let $L\colon \mathbf{Set} \to \mathbf{Mon}$ and $R\colon \mathbf{Mon} \to \mathbf{Set}$ be the free and forgetful functors from Proposition 5.1.1.2. We know that L is left adjoint to R; however L is *not* right adjoint to R. In other words, we can show that the necessary natural isomorphism cannot exist.

Let $X = \{a,b\}$ and let $\mathcal{M} = (\{1\},1,!)$ be the trivial monoid. Then the necessary natural isomorphism would need to give us a bijection

$$\mathrm{Hom}_{\mathbf{Mon}}(\mathcal{M},L(X)) \cong^{?} \mathrm{Hom}_{\mathbf{Set}}(\{1\},X).$$

But the left-hand side has one element, because $\mathcal{M}$ is the initial object in **Mon** (see Example 4.5.3.8), whereas the right-hand side has two elements. Therefore no isomorphism can exist.

Example 5.1.1.6. Preorders have underlying sets, giving rise to a functor $U\colon \mathbf{PrO} \to \mathbf{Set}$. The functor U has both a left adjoint and a right adjoint. The left adjoint of U is $D\colon \mathbf{Set} \to \mathbf{PrO}$, sending a set X to the discrete preorder on X (the preorder with

underlying set X, having the fewest possible $\leqslant$'s). The right adjoint of U is $I\colon \mathbf{Set} \to \mathbf{PrO}$, sending a set X to the indiscrete preorder on X (the preorder with underlying set X, having the most possible $\leqslant$'s). See Example 3.4.4.5.

Exercise 5.1.1.7. Let $U\colon \mathbf{Grph} \to \mathbf{Set}$ denote the functor sending a graph to its underlying set of vertices. This functor has both a left and a right adjoint.

a.) What functor $\mathbf{Set} \to \mathbf{Grph}$ is the left adjoint of U?

b.) What functor $\mathbf{Set} \to \mathbf{Grph}$ is the right adjoint of U?

◊

Example 5.1.1.8. Here are some other adjunctions:

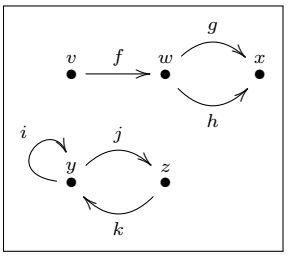
- Ob: $\mathbf{Cat} \to \mathbf{Set}$ has a left adjoint $\mathbf{Set} \to \mathbf{Cat}$ given by the discrete category.
- Ob: $\mathbf{Cat} \to \mathbf{Set}$ has a right adjoint $\mathbf{Set} \to \mathbf{Cat}$ given by the indiscrete category.
- The underlying graph functor $\mathbf{Cat} \to \mathbf{Grph}$ has a left adjoint $\mathbf{Grph} \to \mathbf{Cat}$ given by the free category.
- The functor $\mathbf{PrO} \to \mathbf{Grph}$, given by drawing edges for $\leqslant$'s, has a left adjoint given by existence of paths.
- The forgetful functor from posets to preorders has a left adjoint given by quotient by isomorphism relation.
- Given a set A, the functor $(- \times A)\colon \mathbf{Set} \to \mathbf{Set}$ has a right adjoint $\mathrm{Hom}(A, -)$ (this was called currying in Section 2.7.2).

Exercise 5.1.1.9. Let $F\colon \mathcal{C} \to \mathcal{D}$ and $G\colon \mathcal{D} \to \mathcal{C}$ be mutually inverse equivalences of categories (see Definition 4.3.4.1). Are they adjoint in one direction or the other? ◊

Exercise 5.1.1.10. The discrete category functor $Disc\colon \mathbf{Set} \to \mathbf{Cat}$ has a left adjoint $p\colon \mathbf{Cat} \to \mathbf{Set}$.

a.) For an arbitrary object $X \in \mathrm{Ob}(\mathbf{Set})$ and an arbitrary object $\mathcal{C} \in \mathrm{Ob}(\mathbf{Cat})$, write down the adjunction isomorphism.

b.) Let $\mathcal{C}$ be the free category on the graph G:

$G :=$ 

and let $X = \{1, 2, 3\}$. How many elements does the set $\mathrm{Hom}_{\mathbf{Set}}(\mathcal{C}, Disc(X))$ have?

c.) What can you do to an arbitrary category $\mathcal{C}$ to make a set $p(\mathcal{C})$ such that the adjunction isomorphism holds? That is, how does the functor p behave on objects?

◊

The following proposition says that all adjoints to a given functor are isomorphic to each other.

Proposition 5.1.1.11. *Let $\mathcal{C}$ and $\mathcal{D}$ be categories, let $F\colon \mathcal{C} \to \mathcal{D}$ be a functor, and let $G, G'\colon \mathcal{D} \to \mathcal{C}$ also be functors. If both G and G' are right adjoint (respectively left adjoint) to F then there is a natural isomorphism $\phi\colon G \to G'$.*

Proof. Suppose that both G and G' are right adjoint to F (the case of G and G' being left adjoint is similarly proved). We first give a formula for the components of $\phi\colon G \to G'$ and its inverse $\psi\colon G' \to G$. Given an object $d \in \mathrm{Ob}(\mathcal{D})$, we use $c = G(d)$ to obtain two natural isomorphisms, one from each adjunction:

$$\mathrm{Hom}_{\mathcal{C}}(G(d), G(d)) \cong \mathrm{Hom}_{\mathcal{D}}(F(G(d)), d) \cong \mathrm{Hom}_{\mathcal{C}}(G(d), G'(d)).$$

The identity component $\mathrm{id}_{G(d)}$ is then sent to some morphism $G(d) \to G'(d)$, which we take to be ϕ_d. Similarly, we use $c' = G'(d)$ to obtain two natural isomorphisms, one from each adjunction:

$$\mathrm{Hom}_{\mathcal{C}}(G'(d), G'(d)) \cong \mathrm{Hom}_{\mathcal{D}}(F(G'(d)), d) \cong \mathrm{Hom}_{\mathcal{C}}(G'(d), G(d)).$$

Again, the identity component $\mathrm{id}_{G'(d)}$ is sent to some morphism $G'(d) \to G(d)$, which we take to be ψ_d. The naturality of the isomorphisms implies that ϕ and ψ are natural transformations, and it is straightforward to check that they are mutually inverse.

□

5.1.1.12 Quantifiers as adjoints

One of the simplest but neatest places that adjoints show up is between preimages and the logical quantifiers $\exists$ and $\forall$, which we first discussed in Notation 2.1.1.1. The setting in which to discuss this is that of sets and their power preorders. That is, if X is a set then recall from Section 3.4.2 that the power set $\mathbb{P}(X)$ has a natural ordering by inclusion of subsets.

Given a function $f\colon X \to Y$ and a subset $V \subseteq Y$ the preimage is $f^{-1}(V) := \{x \in X \mid f(x) \in V\}$. If $V' \subseteq V$ then $f^{-1}(V') \subseteq f^{-1}(V)$, so in fact $f^{-1}\colon \mathbb{P}(Y) \to \mathbb{P}(X)$ can be considered a functor (where of course we are thinking of preorders as categories). The quantifiers appear as adjoints of f^{-1}.

Let's begin with the left adjoint of $f^{-1}\colon \mathbb{P}(Y) \to \mathbb{P}(X)$. It is a functor $L_f\colon \mathbb{P}(X) \to \mathbb{P}(Y)$. Choose an object $U \subseteq X$ in $\mathbb{P}(X)$. It turns out that

$$L_f(U) = \{y \in Y \mid \exists x \in f^{-1}(y) \text{ such that } x \in U\}.$$

And the right adjoint $R_f\colon \mathbb{P}(X) \to \mathbb{P}(Y)$, when applied to U is

$$R_f(U) = \{y \in Y \mid \forall x \in f^{-1}(y), x \in U\}.$$

In fact, the functor L_f is generally denoted $\exists_f\colon \mathbb{P}(X) \to \mathbb{P}(Y)$, and R_f is generally denoted $\forall_f\colon \mathbb{P}(X) \to \mathbb{P}(Y)$.

$$\mathbb{P}(X) \underset{\forall_f}{\overset{\exists_f}{\underset{\longleftarrow}{\overset{\longrightarrow}{f^{-1}}}}} \mathbb{P}(Y).$$

We will see in the next example why this notation is apt.

Example 5.1.1.13. In logic or computer science, the quantifiers $\exists$ and $\forall$ are used to ask whether any or all elements of a set have a certain property. For example, one may have a set of natural numbers and want to know whether any or all are even or odd. Let $Y = \{\texttt{even}, \texttt{odd}\}$, and let $p\colon \mathbb{N} \to Y$ be the function that takes assigns to each natural number its parity (even or odd). Because the elements of $\mathbb{P}(\mathbb{N})$ and $\mathbb{P}(Y)$ are ordered by "inclusion of subsets", we can construe these orders as categories (by Proposition 4.2.1.17). That's all old; what's new is that we have adjunctions between these categories

$$\mathbb{P}(\mathbb{N}) \underset{\forall_p}{\overset{\exists_p}{\underset{\longleftarrow}{\overset{\longrightarrow}{p^{-1}}}}} \mathbb{P}(Y).$$

Given a subset $U \subseteq \mathbb{N}$, i.e. an object $U \in \mathrm{Ob}(\mathbb{P}(\mathbb{N}))$, we investigate the objects $\exists_p(U), \forall_p(U)$. These are both subsets of $\{\texttt{even}, \texttt{odd}\}$. The set $\exists_p(U)$ includes the element `even` if there exists an even number in U; it includes the element `odd` if there exists an odd number in U. Similarly, the set $\forall_p(U)$ includes the element `even` if every even number is in U and it includes `odd` if every odd number is in U. [5]

We explain just one of these in terms of the definitions. Let $V = \{\texttt{even}\} \subseteq Y$. Then $f^{-1}(V) \subseteq \mathbb{N}$ is the set of even numbers, and there is a morphism $f^{-1}(V) \to U$ in $\mathbb{P}(\mathbb{N})$ if and only if U contains all the even numbers. Therefore, the adjunction isomorphism $\mathrm{Hom}_{\mathbb{P}(\mathbb{N})}(f^{-1}(V), U) \cong \mathrm{Hom}_{\mathbb{P}(Y)}(V, \forall_p U)$ says that $V \subseteq \forall_p U$, i.e. $\forall_p(U)$ includes the element `even` if and only if U contains all the even numbers, as we said above.

Exercise 5.1.1.14. The national Scout jamboree is a gathering of Boy Scouts from troops across the US. Let X be the set of Boy Scouts in the US, and let Y be the set of Boy Scout troops in the US. Let $t\colon X \to Y$ be the function that assigns to each Boy Scout his troop. Let $U \subseteq X$ be the set of Boy Scouts in attendance at this years jamboree. What is the meaning of the objects $\exists_t U$ and $\forall_t U$? ◊

Exercise 5.1.1.15. Let X be a set and $U \subseteq X$ a subset. Find a set Y and a function $f\colon X \to Y$ such that $\exists_f(U)$ somehow tells you whether U is non-empty, and such that $\forall_f(U)$ somehow tells you whether $U = X$. ◊

In fact, "quantifiers as adjoints" is part of a larger story. Suppose we think of elements of a set X as bins, or storage areas. An element of $\mathbb{P}(X)$ can be construed as an injection $U \hookrightarrow X$, i.e. an assignment of a bin to each element of U, with at most one element of U in each bin. Relaxing that restriction, we may consider arbitrary sets U and assignments $U \to X$ of a bin to each element $u \in U$. Given a function $f\colon X \to Y$, we can generalize $\exists_f$ and $\forall_f$ to functors denoted Σ_f and Π_f, which will parameterize disjoint unions and products (respectively) over $y \in Y$. This will be discussed in Section 5.1.4.

5.1.2 Universal concepts in terms of adjoints

In this section we discuss how universal concepts, i.e. initial objects and terminal objects, colimits and limits, are easily phrased in the language of adjoint functors. We will say that a functor $F\colon \mathcal{C} \to \mathcal{D}$ *is a left adjoint* if there exists a functor $G\colon \mathcal{D} \to \mathcal{C}$ such that F is a left adjoint of G. We showed in Proposition 5.1.1.11 that if F is a left adjoint of some functor G, then it is isomorphic to every other left adjoint of G, and G is isomorphic to every other right adjoint of F.

[5] It may not be clear that by this point we have also handled the question, "is every element of U even?" One simply checks that `odd` is not an element of $\exists_p U$.

Example 5.1.2.1. Let $\mathcal{C}$ be a category and $t\colon \mathcal{C} \to \underline{1}$ the unique functor to the terminal category. Then t is a left adjoint if and only if $\mathcal{C}$ has a terminal object, and t is a right adjoint if and only if $\mathcal{C}$ has an initial object. The proofs are dual, so let's focus on the first.

The functor t has a right adjoint $R\colon \underline{1} \to \mathcal{C}$ if and only if there is an isomorphism

$$\mathrm{Hom}_{\mathcal{C}}(c, r) \cong \mathrm{Hom}_{\underline{1}}(t(c), 1),$$

where $r = R(1)$. But $\mathrm{Hom}_{\underline{1}}(t(c), 1)$ has one element. Thus t has a right adjoint iff there is a unique morphism $c \to r$ in $\mathcal{C}$. This is the definition of r being a terminal object.

When we defined colimits and limits in Definitions 4.5.3.26 and 4.5.3.19 we did so for individual I-shaped diagrams $X\colon I \to \mathcal{C}$. Using adjoints we can define the limit of every I-shaped diagram in $\mathcal{C}$ at once.

Let $t\colon \mathcal{C} \to \underline{1}$ denote the unique functor to the terminal category. Given an object $c \in \mathrm{Ob}(\mathcal{C})$, consider it as a functor $c\colon \underline{1} \to \mathcal{C}$. Then $c \circ t\colon I \to \mathcal{C}$ is the *constant functor at* c, sending each object in I to the same $\mathcal{C}$-object c, and every morphism in I to id_c. This induces a functor that we denote by $\Delta_t\colon \mathcal{C} \to \mathrm{Fun}(I, \mathcal{C})$.

Suppose we want to take the colimit or limit of X. We are given an object X of $\mathrm{Fun}(I, \mathcal{C})$ and we want back an object of $\mathcal{C}$. We could hope, and it turns out to be true, that the adjoints of Δ_t are the limit and colimit. Indeed let $\Sigma_t\colon \mathrm{Fun}(I, \mathcal{C}) \to \mathcal{C}$ be the left adjoint of Δ_t, and let $\Pi_t\colon \mathrm{Fun}(I, \mathcal{C}) \to \mathcal{C}$ be the right adjoint of Δ_t. Then Σ_t is the functor that takes colimits, and Π_t is the functor that takes limits.

We will work with a generalization of colimits and limits in Section 5.1.4. But for now, let's bring this down to earth with a concrete example.

Example 5.1.2.2. Let $\mathcal{C} = \mathbf{Set}$, and let $I = \underline{3}$. The category $\mathrm{Fun}(I, \mathbf{Set})$ is the category of $\{1, 2, 3\}$-indexed sets, e.g. $(\mathbb{Z}, \mathbb{N}, \mathbb{Z}) \in \mathrm{Ob}(\mathrm{Fun}(I, \mathbf{Set}))$ is an object of it. The functor $\Delta_t\colon \mathbf{Set} \to \mathrm{Fun}(I, \mathbf{Set})$ acts as follows. Given a set $c \in \mathrm{Ob}(\mathbf{Set})$, consider it as a functor $c\colon \underline{1} \to \mathbf{Set}$, and the composite $c \circ t\colon I \to \mathbf{Set}$ is the constant functor. That is, $\Delta_t(c)\colon I \to \mathbf{Set}$ is the $\{1, 2, 3\}$–indexed set (c, c, c).

To say that Δ_t has a right adjoint called $\Pi_t\colon \mathrm{Fun}(I, \mathbf{Set}) \to \mathbf{Set}$ and that it "takes limits" should mean that if we look through the definition of right adjoint, we will see that the formula will somehow yield the appropriate limit. Fix a functor $D\colon I \to \mathbf{Set}$, so $D(1), D(2)$, and $D(3)$ are sets. The limit $\lim D$ of D is the product $D(1) \times D(2) \times D(3)$. For example, if $D = (\mathbb{Z}, \mathbb{N}, \mathbb{Z})$ then $\lim D = \mathbb{Z} \times \mathbb{N} \times \mathbb{Z}$. How does this fact arise in the definition of adjoint?

The definition of Π_t being the right adjoint to Δ_t says that there is a natural isomorphism of sets,

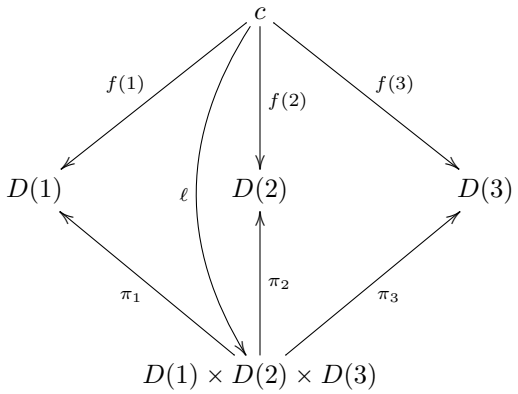
$$\mathrm{Hom}_{\mathrm{Fun}(I, \mathbf{Set})}(\Delta_t(c), D) \cong \mathrm{Hom}_{\mathbf{Set}}(c, \Pi_t(D)). \tag{5.2}$$

The left-hand side has elements $f \in \mathrm{Hom}_{\mathrm{Fun}(I, \mathbf{Set})}(\Delta_t(c), D)$ that look like the left below, but having these three maps is equivalent to having the diagram to the right below:

The isomorphism in (5.2) says that choosing the three maps $f(1), f(2), f(3)$ is the same thing as choosing a function $c \to \Pi_t(D)$. But this is very close to the universal property of limits: there is a unique map $\ell\colon c \to D(1) \times D(2) \times D(3)$, so this product serves well as Π_t as we have said. We're not giving a formal proof here, but what is missing at this point is the fact that certain diagrams have to commute. This comes down to the naturality of the isomorphism (5.2). The map ℓ induces a naturality square

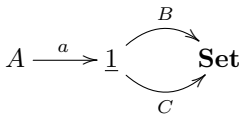
$$\begin{array}{ccc} \Delta_t(c) & \xrightarrow{\Delta_t(\ell)} & \Delta_t\Pi_t D \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \pi} \\ D & = & D \end{array}$$

which says that the following diagram commutes:



It is not hard to show that the composition of left adjoints is a left adjoint, and the composition of right adjoints is a right adjoint. In the following example we show how currying (as in Sections 2.7.2 and 5.1.1.8) arises out of a certain combination of data migration functors.

Example 5.1.2.3 (Currying via Δ, Σ, Π). Let A, B, and C be sets. Consider the unique functor $a\colon A \to \underline{1}$ and consider B and C as functors $\underline{1} \xrightarrow{B}$ **Set** and $\underline{1} \xrightarrow{C}$ **Set** respectively.



Note that $\underline{1}$–**Set** $\cong$ **Set**, and we will elide the difference. Our goal is to see currying arise out of the adjunction between $\Sigma_a \circ \Delta_a$ and $\Pi_a \circ \Delta_a$, which tells us that there is an isomorphism

$$\mathrm{Hom}_{\mathbf{Set}}(\Sigma_a\Delta_a(B), C) \cong \mathrm{Hom}_{\mathbf{Set}}(B, \Pi_a\Delta_a(C)). \tag{5.3}$$

By definition, $\Delta_a(B)\colon A \to$ **Set** assigns to each element $a \in A$ the set B. Since Σ_A takes disjoint unions, we have a bijection

$$\Sigma_a(\Delta_a(B)) = \left(\coprod_{a \in A} B \right) \cong A \times B.$$

Similarly $\Delta_a(C)\colon A \to \mathbf{Set}$ assigns to each element $a \in A$ the set C. Since Π_A takes products, we have a bijection

$$\Pi_a(\Delta_a(C)) = \left(\prod_{a\in A} C\right) \cong C^A.$$

The currying isomorphism $\mathrm{Hom}_{\mathbf{Set}}(A \times B, C) \cong \mathrm{Hom}_{\mathbf{Set}}(B, C^A)$ falls out of (5.3).

5.1.3 Preservation of colimits or limits

One useful fact about adjunctions is that left adjoints preserve all colimits and right adjoints preserve all limits.

Proposition 5.1.3.1. *Let $L\colon \mathcal{B} \rightleftarrows \mathcal{A} : R$ be an adjunction. For any indexing category I and functor $D\colon I \to \mathcal{B}$, if D has a colimit in $\mathcal{B}$ then there is a unique isomorphism*

$$L(\operatorname{colim} D) \cong \operatorname{colim}(L \circ D).$$

Similarly, for any $I \in \mathrm{Ob}(\mathbf{Cat})$ and functor $D\colon I \to \mathcal{A}$, if D has a limit in $\mathcal{A}$ then there is a unique isomorphism

$$R(\lim D) \cong \lim(R \circ D).$$

Proof. The proof is simple if one knows the Yoneda lemma (Section 5.2.1.12). I have decided to skip it to keep the book shorter. See [Mac]. □

Example 5.1.3.2. Since $\mathrm{Ob}\colon \mathbf{Cat} \to \mathbf{Set}$ is both a left adjoint and a right adjoint, it must preserve both limits and colimits. This means that if you want to know the set of objects in the fiber product of some categories, you can simply take the fiber product of the set of objects in those categories,

$$\mathrm{Ob}(\mathcal{A} \times_{\mathcal{C}} \mathcal{B}) \cong \mathrm{Ob}(\mathcal{A}) \times_{\mathrm{Ob}(\mathcal{C})} \mathrm{Ob}(\mathcal{B}).$$

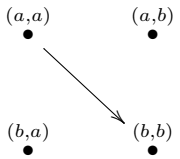
While the right-hand side might look daunting, it is just a fiber product in **Set** which is quite understandable.

This is greatly simplifying. If one thinks through what defines a limit in **Cat**, one is dragged through notions of slice categories and terminal objects in them. These slice categories are in **Cat** so they involve several categories and functors, and it gets hairy or even hopeless to a beginner. Knowing that the objects are given by a simple fiber product makes the search for limits in **Cat** much simpler.

For example, if $[n]$ is the linear order category of length n then $[n] \times [m]$ has $nm + n + m + 1$ objects because $[n]$ has $n + 1$ objects and $[m]$ has $m + 1$ objects.

Example 5.1.3.3. The "path poset" functor $L\colon \mathbf{Grph} \to \mathbf{PrO}$ given by existence of paths (see Exercise 4.1.2.11) is left adjoint to the functor $R\colon \mathbf{PrO} \to \mathbf{Grph}$ given by replacing $\leqslant$'s by arrows. This means that L preserves colimits. So taking the union of graphs G and H results in a graph whose path poset $L(G \sqcup H)$ is the union of the path posets of G and H. But this is not so for products.

Let $G = H = \boxed{a \bullet \xrightarrow{f} \bullet b}$. Then $L(G) = L(H) = [1]$, the linear order of length 1. But the product $G \times H$ in **Grph** looks like the graph



Its preorder $L(G \times H)$ does not have $(a, a) \leqslant (a, b)$, whereas this is the case in $L(G) \times L(H)$.

5.1.4 Data migration

As we saw in Sections 4.2.2 and 4.2.2.5, a database schema is a category $\mathcal{C}$ and an instance is a functor $I : \mathcal{C} \to \mathbf{Set}$.

Notation 5.1.4.1. Let $\mathcal{C}$ be a category. Throughout this section we denote by $\mathcal{C}$–**Set** the category $\mathrm{Fun}(\mathcal{C}, \mathbf{Set})$ of functors from $\mathcal{C}$ to **Set**, i.e. the category of instances on $\mathcal{C}$.

In this section we discuss what happens to the resulting instances when different schemas are connected by a functor, say $F : \mathcal{C} \to \mathcal{D}$. It turns out that three adjoint functors emerge: $\Delta_F : \mathcal{D}$–**Set** $\to \mathcal{C}$–**Set**, $\Sigma_F : \mathcal{C}$–**Set** $\to \mathcal{D}$–**Set**, and $\Pi_F : \mathcal{C}$–**Set** $\to \mathcal{D}$–**Set**, where Δ_F is adjoint to both,

$$\Sigma_F : \mathcal{C}\text{–}\mathbf{Set} \rightleftarrows \mathcal{D}\text{–}\mathbf{Set} : \Delta_F \qquad\qquad \Delta_F : \mathcal{D}\text{–}\mathbf{Set} \rightleftarrows \mathcal{C}\text{–}\mathbf{Set} : \Pi_F.$$

It turns out that almost all the basic database operations are captured by these three functors. For example, Δ_F handles the job of duplicating or deleting tables, as well as duplicating or deleting columns in a single table. The functor Σ_F handles taking unions, and the functor Π_F handles joining tables together, matching columns, or selecting the rows with certain properties (e.g. everyone whose first name is Mary).

5.1.4.2 Pullback: Δ

Given a functor $F : \mathcal{C} \to \mathcal{D}$ and a functor $I : \mathcal{D} \to \mathbf{Set}$, we can compose them to get a functor $I \circ F : \mathcal{C} \to \mathbf{Set}$. In other words, the presence of F provides a way to convert $\mathcal{D}$-instances into $\mathcal{C}$-instances. In fact this conversion is functorial, meaning that morphisms of $\mathcal{D}$-instances are sent to morphisms of $\mathcal{C}$-instances. We denote the resulting functor by $\Delta_F : \mathcal{D}$–**Set** $\to \mathcal{C}$–**Set** and call it *pullback along* F.

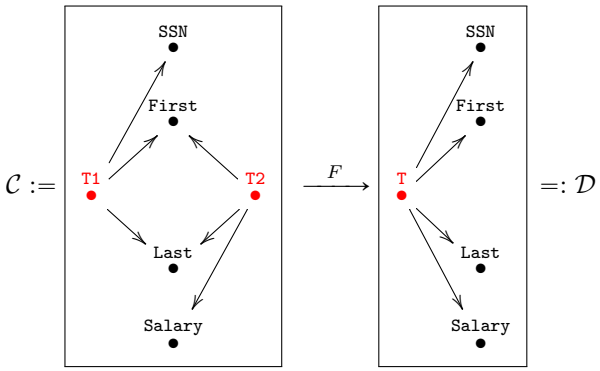
We have seen an example of this before in Example 4.3.2.15, where we showed how a monoid homomorphism $F : \mathcal{M}' \to \mathcal{M}$ could add functionality to a finite state machine. More generally, we can use pullbacks to reorganize data, copying and deleting tables and columns.

Remark 5.1.4.3. Given a functor $F : \mathcal{C} \to \mathcal{D}$, which we think of as a schema translation, the functor $\Delta_F : \mathcal{D}$–**Set** $\to \mathcal{C}$–**Set** "goes the opposite way". The reasoning is simple to any explain (composition of functors) but something about it is often very strange to people, at first. The rough idea of this "contravariance" is captured by the role-reversal in the following slogan:

Slogan 5.1.4.4.

" If I get my information from you, then your information becomes my information. "

Consider the following functor $F\colon \mathcal{C} \to \mathcal{D}$: [6]



(5.4)

Let's spend a moment recalling how to "read" schemas. In schema $\mathcal{C}$ there are leaf tables `SSN, First, Last, Salary`, which represent different kinds of basic data. More interestingly, there are two *fact tables*. The first is called `T1` and it relates `SSN, First,` and `Last`. The second is called `T2` and it relates `First, Last,` and `Salary`.

The functor $F\colon \mathcal{C} \to \mathcal{D}$ relates $\mathcal{C}$ to a schema with a single fact table relating all four attributes: `SSN, First, Last,` and `Salary`. We are interested in $\Delta_F\colon \mathcal{D}$–**Set** $\to \mathcal{C}$–**Set**. Suppose given the following database instance $I\colon \mathcal{D} \to$ **Set** on $\mathcal{D}$:

T				
ID	**SSN**	**First**	**Last**	**Salary**
XF667	115-234	Bob	Smith	$250
XF891	122-988	Sue	Smith	$300
XF221	198-877	Alice	Jones	$100

SSN
ID
115-234
118-334
122-988
198-877
342-164

First
ID
Adam
Alice
Bob
Carl
Sam
Sue

Last
ID
Jones
Miller
Pratt
Richards
Smith

Salary
ID
$100
$150
$200
$250
$300

How do you get the instance $\Delta_F(I)\colon \mathcal{C} \to$ **Set**? The formula was given above: compose I with F. In terms of tables, it feels like duplicating table `T` as `T1` and `T2`, but deleting a column from each in accordance with the definition of $\mathcal{C}$ in (5.4). Here is the result, $\Delta_F(I)$, in table form:

[6] This example was taken from [Sp1], http://arxiv.org/abs/1009.1166.

T1			
ID	**SSN**	**First**	**Last**
XF667	115-234	Bob	Smith
XF891	122-988	Sue	Smith
XF221	198-877	Alice	Jones

T2			
ID	**First**	**Last**	**Salary**
XF221	Alice	Jones	$100
XF667	Bob	Smith	$250
XF891	Sue	Smith	$300

SSN
ID
115-234
118-334
122-988
198-877
342-164

First
ID
Adam
Alice
Bob
Carl
Sam
Sue

Last
ID
Jones
Miller
Pratt
Richards
Smith

Salary
ID
$100
$150
$200
$250
$300

Exercise 5.1.4.5. Let $\mathcal{C} = (G, \simeq)$ be a schema. A leaf table is an object $c \in \mathrm{Ob}(\mathcal{C})$ with no outgoing arrows.

a.) Write the condition of being a "leaf table" mathematically in three different languages: that of graphs (using symbols V, A, src, tgt), that of categories (using $\mathrm{Hom}_{\mathcal{C}}$, etc.), and that of tables (in terms of columns, tables, rows, etc.).

b.) In the language of categories, is there a difference between a terminal object and a leaf table? Explain.

◊

Exercise 5.1.4.6. Consider the schemas

$$[1] = \boxed{0 \bullet \xrightarrow{f} \bullet 1} \qquad \text{and} \qquad [2] = \boxed{0 \bullet \xrightarrow{g} \bullet 1 \xrightarrow{h} \bullet 2},$$

and the functor $F\colon [1] \to [2]$ given by sending $0 \mapsto 0$ and $1 \mapsto 2$.

a.) How many possibilities are there for $F(f)$?

b.) Now suppose $I\colon [2] \to \mathbf{Set}$ is given by the following tables.

0	
ID	**g**
Am	To be verb
Baltimore	Place
Carla	Person
Develop	Action verb
Edward	Person
Foolish	Adjective
Green	Adjective

1	
ID	**h**
Action verb	Verb
Adjective	Adjective
Place	Noun
Person	Noun
To be verb	Verb

2
ID
Adjective
Noun
Verb

Write out the two tables associated to the $[1]$-instance $\Delta_F(I)\colon [1] \to \mathbf{Set}$.

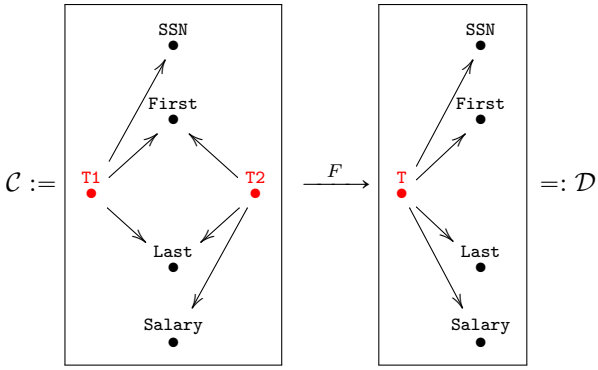
◊

5.1.4.7 Left pushforward: Σ

Let $F\colon \mathcal{C} \to \mathcal{D}$ be a functor. The functor $\Delta_F\colon \mathcal{D}$–**Set** $\to \mathcal{C}$–**Set** has a left adjoint, $\Sigma_F\colon \mathcal{C}$–**Set** $\to \mathcal{D}$–**Set**. The rough idea is that Σ_F performs parameterized colimits. Given an instance $I\colon \mathcal{C} \to$ **Set**, we get an instance on $\mathcal{D}$ that acts as follows. For each object $d \in \mathrm{Ob}(\mathcal{D})$, the set $\Sigma_F(I)(d)$ is the colimit (think, union) of some diagram back home in $\mathcal{C}$.

Left pushforwards (also known as left Kan extensions) are discussed at length in [Sp1]; here we begin with some examples from that paper.

Example 5.1.4.8. We again use the functor $F\colon \mathcal{C} \to \mathcal{D}$ drawn below



(5.4)

We will be applying the left pushforward $\Sigma_F\colon \mathcal{C}$–**Set** $\to \mathcal{D}$–**Set** to the following instance $I\colon \mathcal{C} \to$ **Set**:

T1			
ID	**SSN**	**First**	**Last**
T1-001	115-234	Bob	Smith
T1-002	122-988	Sue	Smith
T1-003	198-877	Alice	Jones

T2			
ID	**First**	**Last**	**Salary**
T2-001	Alice	Jones	$100
T2-002	Sam	Miller	$150
T2-004	Sue	Smith	$300
T2-010	Carl	Pratt	$200

SSN
ID
115-234
118-334
122-988
198-877
342-164

First
ID
Adam
Alice
Bob
Carl
Sam
Sue

Last
ID
Jones
Miller
Pratt
Richards
Smith

Salary
ID
$100
$150
$200
$250
$300

The functor $F\colon \mathcal{C} \to \mathcal{D}$ sent both tables `T1` and `T2` to table `T`. Applying Σ_F will take the what was in `T1` and `T2` and put the union in `T`. The result $\Sigma_F I\colon \mathcal{D} \to$ **Set** is as follows:

T				
ID	**SSN**	**First**	**Last**	**Salary**
T1-001	115-234	Bob	Smith	T1-001.Salary
T1-002	122-988	Sue	Smith	T1-002.Salary
T1-003	198-877	Alice	Jones	T1-003.Salary
T2-001	T2-A101.SSN	Alice	Jones	$100
T2-002	T2-A102.SSN	Sam	Miller	$150
T2-004	T2-004.SSN	Sue	Smith	$300
T2-010	T2-A110.SSN	Carl	Pratt	$200

SSN
ID
115-234
118-334
122-988
198-877
342-164
T2-001.SSN
T2-002.SSN
T2-004.SSN
T2-010.SSN

First
ID
Adam
Alice
Bob
Carl
Sam
Sue

Last
ID
Jones
Miller
Pratt
Richards
Smith

Salary
ID
$100
$150
$200
$250
$300
T1-001.Salary
T1-002-Salary
T1-003-Salary

As you can see, there was no set salary information for any data coming from table `T1` nor any set SSN information for any data coming form table `T2`. But the definition of adjoint, given in Definition 5.1.1.1, yielded the universal response: freely add new variables that take the place of missing information. It turns out that this idea already has a name in logic, *Skolem variables*, and a name in database theory, *labeled nulls*.

Exercise 5.1.4.9. Consider the functor $F\colon \underline{3} \to \underline{2}$ sending $1 \mapsto 1, 2 \mapsto 2, 3 \mapsto 2$.

a.) Write down an instance $I\colon \underline{3} \to \mathbf{Set}$.

b.) Given the description that "Σ_F performs a parameterized colimit", make an educated guess about what $\Sigma_F(I)$ will be. Give your answer in the form of two sets that are made up from the three sets you already wrote down.

$\Diamond$

We now briefly give the actual formula for computing left pushforwards. Suppose that $F\colon \mathcal{C} \to \mathcal{D}$ is a functor and let $I\colon \mathcal{C} \to \mathbf{Set}$ be a set-valued functor on $\mathcal{C}$. Then $\Sigma_F(I)\colon \mathcal{D} \to \mathbf{Set}$ is defined as follows. Given an object $d \in \mathrm{Ob}(\mathcal{D})$ we first form the comma category (see Definition 4.6.4.1) for the setup

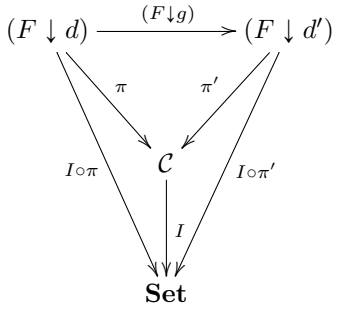
$$\mathcal{C} \xrightarrow{F} \mathcal{D} \xleftarrow{d} \underline{1}$$

and denote it by $(F \downarrow d)$. There is a canonical projection functor $\pi\colon (F \downarrow d) \to \mathcal{C}$, which we can compose with $I\colon \mathcal{C} \to \mathbf{Set}$ to obtain a functor $(F \downarrow d) \to \mathbf{Set}$. We are ready to define $\Sigma_F(I)(d)$ to be its colimit,

$$\Sigma_F(I)(d) := \operatorname*{colim}_{(F\downarrow d)} I \circ \pi.$$

We have defined $\Sigma_F(I)\colon \mathcal{D} \to \mathbf{Set}$ on objects $d \in \mathrm{Ob}(\mathcal{D})$. As for morphisms we will be even more brief, but one can see [Sp1] for details. Given a morphism $g\colon d \to d'$ one

notes that there is an induced functor $(F \downarrow g)\colon (F \downarrow d) \to (F \downarrow d')$ and a commutative diagram of categories:



By the universal property of colimits, this induces the required function

$$\operatorname*{colim}_{(F\downarrow d)} I \circ \pi \xrightarrow{\Sigma_F(I)(g)} \operatorname*{colim}_{(F\downarrow d')} I \circ \pi'.$$

5.1.4.10 Right pushforward: Π

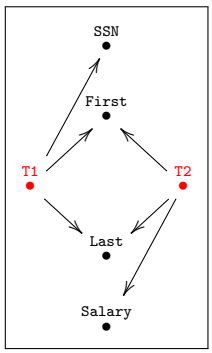
Let $F\colon \mathcal{C} \to \mathcal{D}$ be a functor. We heard in Section 5.1.4.7 that the functor $\Delta_F\colon \mathcal{D}$–**Set** $\to$ $\mathcal{C}$–**Set** has a left adjoint. Here we explain that it has a right adjoint, $\Pi_F\colon \mathcal{C}$–**Set** $\to \mathcal{D}$–**Set** as well. The rough idea is that Π_F performs parameterized limits. Given an instance $I\colon \mathcal{C} \to$ **Set**, we get an instance on $\mathcal{D}$ that acts as follows. For each object $d \in \mathrm{Ob}(\mathcal{D})$, the set $\Pi_F(I)(d)$ is the limit (think, fiber product) of some diagram back home in $\mathcal{C}$.

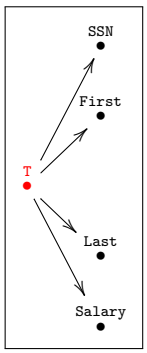
Right pushforwards (also known as right Kan extensions) are discussed at length in [Sp1]; here we begin with some examples from that paper.

Example 5.1.4.11. We once again use the functor $F\colon \mathcal{C} \to \mathcal{D}$ from Example 5.1.4.8. We will apply the right pushforward Π_F to instance $I\colon \mathcal{C} \to$ **Set** from that example. [7]

The instance $\Pi_F(I)$ will put data in all 5 tables in $\mathcal{D}$. In `T` it will put pairs (t_1, t_2) where t_1 is a row in `T1` and t_2 is a row in `T2` for which the first and last names agree.

[7] To repeat for convenience,

$\mathcal{C} :=$ 

$\xrightarrow{F}$ 

$=: \mathcal{D}$ (5.4)

$I\colon \mathcal{C} \to$ **Set** is

T1			
ID	**SSN**	**First**	**Last**
T1-001	115-234	Bob	Smith
T1-002	122-988	Sue	Smith
T1-003	198-877	Alice	Jones

T2			
ID	**First**	**Last**	**Salary**
T2-001	Alice	Jones	$100
T2-002	Sam	Miller	$150
T2-004	Sue	Smith	$300
T2-010	Carl	Pratt	$200

It will copy the leaf tables exactly, so we do not display them here; the following is the table T for $\Pi_F(I)$:

T				
ID	**SSN**	**First**	**Last**	**Salary**
T1-002T2-A104	122-988	Sue	Smith	$300
T1-003T2-A101	198-877	Alice	Jones	$100

Looking at T1 and T2, there were only two ways to match first and last names.

Exercise 5.1.4.12. Consider the functor $F\colon \underline{3} \to \underline{2}$ sending $1 \mapsto 1, 2 \mapsto 2, 3 \mapsto 2$.

a.) Write down an instance $I\colon \underline{3} \to \mathbf{Set}$.

b.) Given the description that "Π_F performs a parameterized limit", make an educated guess about what $\Pi_F(I)$ will be. Give your answer in the form of two sets that are made up from the three sets you already wrote down.

◊

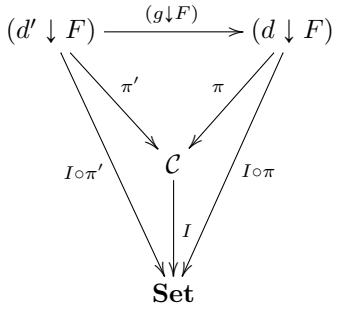
We now briefly give the actual formula for computing right pushforwards. Suppose that $F\colon \mathcal{C} \to \mathcal{D}$ is a functor and let $I\colon \mathcal{C} \to \mathbf{Set}$ be a set-valued functor on $\mathcal{C}$. Then $\Pi_F(I)\colon \mathcal{D} \to \mathbf{Set}$ is defined as follows. Given an object $d \in \mathrm{Ob}(\mathcal{D})$ we first form the comma category (see Definition 4.6.4.1) for the setup

$$\underline{1} \xrightarrow{d} \mathcal{D} \xleftarrow{F} \mathcal{C}$$

and denote it by $(d \downarrow F)$. There is a canonical projection functor $\pi\colon (d \downarrow F) \to \mathcal{C}$, which we can compose with $I\colon \mathcal{C} \to \mathbf{Set}$ to obtain a functor $(d \downarrow F) \to \mathbf{Set}$. We are ready to define $\Pi_F(I)(d)$ to be its limit,

$$\Pi_F(I)(d) := \lim_{(d \downarrow F)} I \circ \pi.$$

We have defined $\Pi_F(I)\colon \mathcal{D} \to \mathbf{Set}$ on objects $d \in \mathrm{Ob}(\mathcal{D})$. As for morphisms we will be even more brief, but one can see [Sp1] for details. Given a morphism $g\colon d \to d'$ one notes that there is an induced functor $(g \downarrow F)\colon (d' \downarrow F) \to (d \downarrow F)$ and a commutative diagram of categories:



SSN
ID
115-234
118-334
122-988
198-877
342-164

First
ID
Adam
Alice
Bob
Carl
Sam
Sue

Last
ID
Jones
Miller
Pratt
Richards
Smith

Salary
ID
$100
$150
$200
$250
$300

By the universal property of limits, this induces the required function

$$\lim_{(d\downarrow F)} I \circ \pi \xrightarrow{\Pi_F(I)(g)} \lim_{(d'\downarrow F)} I \circ \pi'.$$

5.2 Categories of functors

For any two categories $\mathcal{C}$ and $\mathcal{D}$, [8] we discussed the category $\mathrm{Fun}(\mathcal{C},\mathcal{D})$ of functors and natural transformations between them. In this section we discuss functor categories a bit more and give some important applications within mathematics (sheaves) that extend to the real world.

5.2.1 Set-valued functors

Let $\mathcal{C}$ be a category. Then we have been writing $\mathcal{C}$–**Set** to denote the functor category $\mathrm{Fun}(\mathcal{C},\mathbf{Set})$. Here is a nice result about these categories.

Proposition 5.2.1.1. *Let $\mathcal{C}$ be a category. The category $\mathcal{C}$–**Set** is closed under colimits and limits.*

Sketch of proof. Let J be an indexing category and $D\colon J \to \mathcal{C}$–**Set** a functor. For each object $c \in \mathrm{Ob}(\mathcal{C})$, we have a functor $D_c\colon J \to \mathbf{Set}$ defined by $D_c(j) = D(j)(c)$. Define a functor $L\colon \mathcal{C} \to \mathbf{Set}$ by $L(c) = \lim_J D_c$, and note that for each $f\colon c \to c'$ in $\mathcal{C}$ there is an induced function $L(f)\colon L(c) \to L(c')$. One can check that L is a limit of J, because it satisfies the relevant universal property.

The dual proof holds for colimits.

□

Application 5.2.1.2. When taking in data about a scientific subject, one often finds that the way one thinks about the problem changes over time. We understand this phenomenon in the language of databases in terms of a series of schemas $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$, perhaps indexed chronologically. The problem is that old data is held in old schemas and we want to see it in our current understanding. The first step is to transfer all the old data to our new schema in the freest possible way, that is, making no assumptions about how to fill in the new fields. If one creates functors $F_i\colon \mathcal{C}_i \to \mathcal{C}_{i+1}$ from each of these schemas to the next, then we can push the data forward using Σ_{F_i}.

Doing this we will have n datasets on $\mathcal{D} := \mathcal{C}_n$, namely one for each "epoch of understanding". Since the category $\mathcal{D}$–**Set** has all colimits, we can take the union of these datasets and get one. It will have many Skolem variables (see Example 5.1.4.8), and these need to be handled in a coherent way. However, the universality of left adjoints could be interpreted as saying that any reasonable formula for handling this old data can be applied to our results.

◊◊

Exercise 5.2.1.3. By Proposition 5.2.1.1, the category $\mathcal{C}$–**Set** is closed under taking limits. By Exercises 4.5.3.21 and 4.5.3.28, this means in particular that $\mathcal{C}$–**Set** has an initial object and a terminal object.

a.) Let $A \in \mathrm{Ob}(\mathcal{C}\text{–}\mathbf{Set})$ be the initial object, considered as a functor $A\colon \mathcal{C} \to \mathbf{Set}$. For any $c \in \mathrm{Ob}(\mathcal{C})$, what is the set $A(c)$?

[8] Technically $\mathcal{C}$ has to be small (see Remark 4.1.1.2), but as we said there, we are not worrying about that distinction in this book.

b.) Let $Z \in \mathrm{Ob}(\mathcal{C}\text{–}\mathbf{Set})$ be the terminal object, considered as a functor $Z\colon \mathcal{C} \to \mathbf{Set}$. For any $c \in \mathrm{Ob}(\mathcal{C})$, what is the set $Z(c)$?

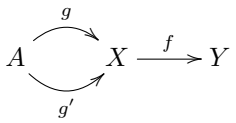
◊

Proposition 5.2.1.1 says that we can add or multiply database states together. In fact, database states on $\mathcal{C}$ form what is called a *topos* which means that just about every consideration we made for sets holds for instances on any schema. Perhaps the simplest schema is $\mathcal{C} = \boxed{\bullet}$, on which the relevant topos is indeed **Set**. But schemas can be arbitrarily complex, and it is impressive that all of these considerations make sense in such generality. Here is a table that makes a comparison between these domains.

Dictionary between **Set** and $\mathcal{C}$–**Set**	
Concept in Set	**Concept in $\mathcal{C}$–Set**
Set	Object in $\mathcal{C}$–**Set**
Function	Morphism in $\mathcal{C}$–**Set**
Element	Representable functor
Empty set	Initial object
Natural numbers	Natural numbers object
Image	Image
(Co)limits	(Co)limits
Exponential objects	Exponential objects
"Familiar" arithmetic	"Familiar" arithmetic
Power sets 2^X	Power objects Ω^X
Characteristic functions	Characteristic morphisms
Surjections, injections	Epimorphisms, monomorphisms

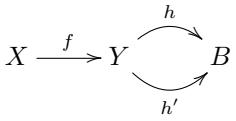
In the above table we said that elements of a set are akin to representable functors in $\mathcal{C}$–**Set**, but we have not yet defined those; we do so in Section 5.2.1.6. First we briefly discuss monomorphisms and epimorphisms in general (Definition 5.2.1.4) and then in $\mathcal{C}$–**Set** (Proposition 5.2.1.5).

Definition 5.2.1.4 (Monomorphism, Epimorphism)**.** Let $\mathcal{S}$ be a category and let $f\colon X \to Y$ be a morphism. We say that f is a *monomorphism* if it has the following property. For all objects $A \in \mathrm{Ob}(\mathcal{S})$ and morphisms $g, g'\colon A \to X$ in $\mathcal{S}$,



if $f \circ g = f \circ g'$ then $g = g'$.

We say that $f\colon X \to Y$ is an *epimorphism* if it has the following property. For all objects $B \in \mathrm{Ob}(\mathcal{S})$ and morphisms $h, h'\colon Y \to B$ in $\mathcal{S}$,



if $h \circ f = h' \circ f$ then $h = h'$.

In the category of sets, monomorphisms are the same as injections and epimorphisms are the same as surjections (see Proposition 2.7.5.4). The same is true in $\mathcal{C}$–**Set**: one can check "table by table" that a morphism of instances is mono or epi.

Proposition 5.2.1.5. *Let $\mathcal{C}$ be a category and let $X, Y : \mathcal{C} \to$ **Set** be objects in $\mathcal{C}$–**Set** and let $f : X \to Y$ be a morphism in $\mathcal{C}$–**Set**. Then f is a monomorphism (respectively an epimorphism) if and only if, for every object $c \in \mathrm{Ob}(\mathcal{C})$, the function $f(c) : X(c) \to Y(c)$ is injective (respectively surjective).*

Sketch of proof. We first show that if f is mono (respectively epi) then so is $f(c)$ for all $c \in \mathrm{Ob}(\mathcal{C})$. Considering c as a functor $c : \underline{1} \to \mathcal{C}$, this result follows from the fact that Δ_c preserves limits and colimits, hence monos and epis.

We now check that if $f(c)$ is mono for all $c \in \mathrm{Ob}(\mathcal{C})$ then f is mono. Suppose that $g, g' : A \to X$ are morphisms in $\mathcal{C}$–**Set** such that $f \circ g = f \circ g'$. Then for every c we have $f \circ g(c) = f \circ g'(c)$ which implies by hypothesis that $g(c) = g'(c)$. But the morphisms in $\mathcal{C}$–**Set** are natural transformations, and if two natural transformations g, g' have the same components then they are the same.

A similar argument works to show the analogous result for epimorphisms.

□

5.2.1.6 Representable functors

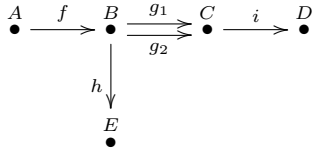
Given a category $\mathcal{C}$, there are certain functors $\mathcal{C} \to$ **Set** that come with the package, one for every object in $\mathcal{C}$. So if $\mathcal{C}$ is a database schema, then for every table $c \in \mathrm{Ob}(\mathcal{C})$ there is a certain database instance associated to it. These instances, i.e. set-valued functors, are called representable functors, and they'll be defined in Definition **??**. The idea is that if a database schema represents a conceptual layout of types (e.g. as an olog), then each type T has an instance associated to it, standing for "the generic thing of type T with all its generic attributes".

Definition 5.2.1.7. Let $\mathcal{C}$ be a category and let $c \in \mathrm{Ob}(\mathcal{C})$ be an object. The functor $\mathrm{Hom}_{\mathcal{C}}(c, -) : \mathcal{C} \to$ **Set**, sending $d \in \mathrm{Ob}(\mathcal{C})$ to the set $\mathrm{Hom}_{\mathcal{C}}(c, d)$ and acting similarly on morphisms $d \to d'$, is said to be *represented by* c. If a functor $F : \mathcal{C} \to$ **Set** is isomorphic to $\mathrm{Hom}_{\mathcal{C}}(c, -)$, we say that F is *a representable functor*. We sometimes write $Y_c := \mathrm{Hom}_{\mathcal{C}}(c, -)$ for short.

Example 5.2.1.8. Given a category $\mathcal{C}$ and an object $c \in \mathrm{Ob}(\mathcal{C})$, we get a representable functor. If we think of $\mathcal{C}$ as a database schema and c as a table, then what does the representable functor $Y_c : \mathcal{C} \to$ **Set** look like in terms of databases? It turns out that the following procedure will generate it.

Begin by writing a new row, say "☺", in the ID column of table c. For each foreign key column $f : c \to c'$, add a row in the ID column of table c' called "f(☺)" and record that result (i.e. "f(☺)") in the f column of table c. Repeat as follows: for each table d, identify all rows r that have blank cell in column $g : d \to e$. Add a new row called "$g(r)$" to table e and record that result in the (r, g) cell of table d.

Here is a concrete example. Let $\mathcal{C}$ be the following schema:



Then $Y_B : \mathcal{C} \to \mathbf{Set}$ is the following instance

A	
ID	f

B			
ID	g_1	g_2	h
☺	$g_1(☺)$	$g_2(☺)$	$h(☺)$

C	
ID	i
$g_1(☺)$	$i(g_1(☺))$
$g_2(☺)$	$i(g_2(☺))$

D
ID
$i(g_1(☺))$
$i(g_2(☺))$

E
ID
$h(☺)$

We began with a single element in table B and followed the arrows, putting new entries wherever they were required. One might call this the *schematically implied reference spread* or *SIRS* of the element ☺ in table B. Notice that the table at A is empty, because there are no morphisms $B \to A$.

Representable functors Y_c yield databases states that are as free as possible, subject to having the initial row ☺ in table c. We have seen things like this before (by the name of Skolem variables) when studying the left pushforward Σ. Indeed, if $c \in \mathrm{Ob}(\mathcal{C})$ is an object, we can consider it as a functor $c : \underline{1} \to \mathcal{C}$. A database instance on $\underline{1}$ is the same thing as a set X. The left pushforward $\Sigma_c(X)$ has the same kinds of Skolem variables. If $X = \{☺\}$ is a one element set, then we get the representable functor $\Sigma_c(\{☺\}) \cong Y_c$.

Exercise 5.2.1.9. Consider the schema for graphs,

$$\mathbf{GrIn} := \boxed{Ar \bullet \underset{tgt}{\overset{src}{\rightrightarrows}} \bullet Ve}$$

a.) Write down the representable functor $Y_{Ar} : \mathbf{GrIn} \to \mathbf{Set}$ as two tables.

b.) Write down the representable functor Y_{Ve} as two tables.

◊

Exercise 5.2.1.10. Consider the loop schema

$$\mathcal{L}oop := \boxed{f \circlearrowright s \bullet}.$$

What is the representable functor $Y_s : \mathcal{L}oop \to \mathbf{Set}$? ◊

Let B be a box in an olog, say ⌜a person⌝, and recall that an aspect of B is an outgoing arrow, such as ⌜a person⌝ $\xrightarrow{\text{has as height in inches}}$ ⌜an integer⌝. The following slogan explains representable functors in those terms.

Slogan 5.2.1.11.

> " *The functor represented by* ⌜a person⌝ *simply leaves a placeholder, like* ⟨*person's name here*⟩ *or* ⟨*person's height here*⟩, *for every aspect of* ⌜a person⌝.
>
> *In general, there is a representable functor for every type in an olog. The representable functor for type T simply encapsulates the most generic or abstract example of type T, by leaving a placeholder for each of its attributes.* "

5.2.1.12 Yoneda's lemma

One of the most powerful tools in category theory is Yoneda's lemma. It is often considered by new students to be quite abstract, but grounding it in databases may help.

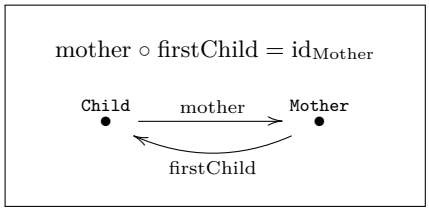
The idea is this. Suppose that $I : \mathcal{C} \to \mathbf{Set}$ is a database instance, and let $c \in \mathrm{Ob}(\mathcal{C})$ be an object. Because I is a functor, we know that for every row $r \in I(c)$ in table c a value has been recorded in the f-column, where $f : c \to c'$ is any outgoing arrow. The value in the (r, f)-cell refers to some row in table c'. What we're saying is that each row in table c induces SIRS throughout the database. They may not be "Skolem", or in any sense "freely generated", but they are there nonetheless. The point is that to each row in c there is a unique mapping $Y_c \to I$.

Lemma 5.2.1.13 (Yoneda's lemma, part 1.)**.** *Let $\mathcal{C}$ be a category, $c \in \mathrm{Ob}(\mathcal{C})$ an object, and $I : \mathcal{C} \to \mathbf{Set}$ a set-valued functor. There is a natural bijection*

$$\mathrm{Hom}_{\mathcal{C}\text{-}\mathbf{Set}}(Y_c, I) \xrightarrow{\cong} I(c).$$

Proof. See [Mac]. □

Example 5.2.1.14. Consider the category $\mathcal{C}$ drawn below:

$\mathcal{C} :=$ 

There are two representable functors, $Y_{\texttt{Child}}$ and $Y_{\texttt{Mother}}$. The latter, when written as a database instance, will consist of a single row in each table. The former, $Y_{\texttt{Child}} : \mathcal{C} \to \mathbf{Set}$ is shown here:

Child	
ID	**mother**
☺	mother(☺)
firstChild(mother(☺))	mother(☺)

Mother	
ID	**firstChild**
mother(☺)	firstChild(mother(☺))

The representable functor $Y_{\texttt{Child}}$ is the freest instance possible, starting with one element in the Child table and satisfying the constraints.

Here is another instance $I : \mathcal{C} \to \mathbf{Set}$:

Child	
ID	**mother**
Amy	Ms. Adams
Bob	Ms. Adams
Carl	Ms. Jones
Deb	Ms. Smith

Mother	
ID	**firstChild**
Ms. Adams	Bob
Ms. Jones	Carl
Ms. Smith	Deb

Yoneda's lemma (5.2.1.13) is about the set of natural transformations $Y_{\texttt{Child}} \to I$. Recall from Definition 4.3.1.2 that a search for natural transformations can get a bit tedious. Yoneda's lemma makes the calculation quite trivial. In our case there are exactly four such natural transformations, and they are completely determined by where ☺ goes. In some sense the symbol ☺ *represents* child-ness in our database.

Exercise 5.2.1.15. Consider the schema $\mathcal{C}$ and instance $I\colon \mathcal{C} \to \mathbf{Set}$ from Example 5.2.1.14. Let $Y_{\texttt{Child}}$ be the representable functor as above.

a.) Let $\alpha\colon Y_{\texttt{Child}} \to I$ be the natural transformation sending ☺ to Amy. What is $\alpha_{\text{Child}}(\text{firstChild}(\text{mother}(☺)))$? [9]

b.) Let $\alpha\colon Y_{\texttt{Child}} \to I$ be the natural transformation sending ☺ to Bob. What is $\alpha_{\texttt{Child}}(\text{firstChild}(\text{mother}(☺)))$?

c.) Let $\alpha\colon Y_{\texttt{Child}} \to I$ be the natural transformation sending ☺ to Carl. What is $\alpha_{\texttt{Child}}(\text{firstChild}(\text{mother}(☺)))$?

d.) Let $\alpha\colon Y_{\texttt{Child}} \to I$ be the natural transformation sending ☺ to Deb. What is $\alpha_{\texttt{Child}}(\text{firstChild}(\text{mother}(☺)))$?

e.) Let $\alpha\colon Y_{\texttt{Child}} \to I$ be the natural transformation sending ☺ to Amy. What is $\alpha_{\texttt{Mother}}(\text{mother}(☺))$?

◊

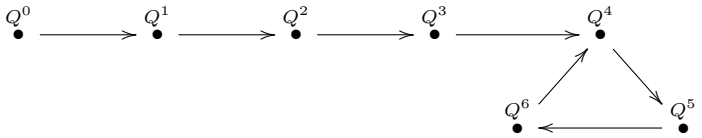
We saw in Section 5.2.1.6 that a representable functor is a mathematically-generated database instance for an abstract thing of type T. It creates placeholders for every attribute that things of type T are supposed to have.

Slogan 5.2.1.16.

> " *Yoneda's lemma says the following. Specifying an actual thing of type T is the same as filling in all placeholders found in the generic thing of type T.* "

Yoneda's lemma is considered by many category theory lovers to be the most important tool in the subject. While its power is probably unclear to students whose sole background in category theory comes from this book, Yoneda's lemma is indeed extremely useful for reasoning. It allows us to move the notion of functor application into the realm of morphisms between functors (i.e. morphisms in $\mathcal{C}$–**Set**, which are natural transformations). This keeps everything in one place — it's all in the morphisms — and thus more interoperable.

Example 5.2.1.17. In Example 3.1.1.26, we discussed the cyclic monoid $\mathcal{M}$ generated by the symbol Q and subject to the relation $Q^7 = Q^4$. We drew a picture like this:



(5.5)

We are finally ready to give the mathematical foundation for this picture. Since $\mathcal{M}$ is a category with one object, ▲, there is a unique representable functor (up to isomorphism) $Y := Y_{▲}\colon \mathcal{M} \to \mathbf{Set}$. A functor $\mathcal{M} \to \mathbf{Set}$ can be thought of as a set with an $\mathcal{M}$-action, as discussed in Section 4.2.1.1. Here the required set is

$$Y(▲) = \text{Hom}_{\mathcal{M}}(▲, ▲) \cong \{Q^0, Q^1, Q^2, Q^3, Q^4, Q^5, Q^6\}$$

[9] There is a lot of clutter, perhaps. Note that "firstChild(mother(☺))" is a row in the `Child` table. Assuming that the math follows the meaning, if ☺ points to Amy, where should firstChild(Mother(☺)) point?

and the action is pretty straightforward (it is called the *principal action*). We might say that (5.5) is a picture of this principal action of $\mathcal{M}$.

However, we can go one step further. Given a functor $Y\colon \mathcal{M} \to \mathbf{Set}$, we can take its category of elements, $\int_{\mathcal{M}} Y$ as in Section 4.6.2. The category $\int_{\mathcal{M}} Y$ has objects $Y(\blacktriangle) \in \mathrm{Ob}(\mathbf{Set})$, i.e. the set of dots in (5.5), and it has a unique morphism $Q^i \to Q^j$ for every path of length $\leqslant 6$ from Q^i to Q^j in that picture.

Exercise 5.2.1.18. Let $c \in \mathrm{Ob}(\mathcal{C})$ be an object and let $I \in \mathrm{Ob}(\mathcal{C}\text{–}\mathbf{Set})$ be another object. Consider c also as a functor $c\colon \underline{1} \to \mathcal{C}$ and recall the pullback functor $\Delta_c\colon \mathcal{C}\text{–}\mathbf{Set} \to \mathbf{Set}$ and its left adjoint $\Sigma_c\colon \mathbf{Set} \to \mathcal{C}\text{–}\mathbf{Set}$ from Section 5.1.4.

a.) What is the set $\Delta_c(I)$?

b.) What is $\mathrm{Hom}_{\mathbf{Set}}(\{\text{☺}\}, \Delta_c(I))$?

c.) What is $\mathrm{Hom}_{\mathcal{C}\text{–}\mathbf{Set}}(\Sigma_c(\{\text{☺}\}), I)$?

d.) How does $\Sigma_c(\{\text{☺}\})$ compare to Y_c, the functor represented by c, as objects in $\mathcal{C}$–**Set**?

◊

Lemma 5.2.1.19 (Yoneda's lemma, part 2)**.** *Let $\mathcal{C}$ be a category. The assignment $c \mapsto Y_c$ from Lemma 5.2.1.13 extends to a functor $Y\colon \mathcal{C}^{\mathrm{op}} \to \mathcal{C}\text{–}\mathbf{Set}$, and this functor is fully faithful.*

*In particular, if $c, c' \in \mathrm{Ob}(\mathcal{C})$ are objects and there is an isomorphism $Y_c \cong Y_{c'}$ in $\mathcal{C}$–**Set**, then there is an isomorphism $c \cong c'$ in $\mathcal{C}$.*

Proof. See [Mac]. □

Exercise 5.2.1.20. The distributive law for addition of natural numbers says $(a+b)\times c = a\times c + b\times c$. Below we will give a proof of the distributive law, using category-theoretic reasoning. Annotate anything in red ink with a justification for why it is true.

Proposition 5.2.1.21. *For any natural numbers $a, b, c \in \mathbb{N}$, the distributive law*

$$(a+b)c = ac + bc$$

holds.

Sketch of proof. To finish, justify red stuff.
Let A, B, C be finite sets and let X be another finite set.

$$\begin{aligned}\mathrm{Hom}_{\mathbf{Set}}((A+B)\times C, X) &\cong \mathrm{Hom}_{\mathbf{Set}}(A+B, X^C)\\ &\cong \mathrm{Hom}_{\mathbf{Set}}(A, X^C) \times \mathrm{Hom}_{\mathbf{Set}}(B, X^C)\\ &\cong \mathrm{Hom}_{\mathbf{Set}}(A\times C, X) \times \mathrm{Hom}_{\mathbf{Set}}(B\times C, X)\\ &\cong \mathrm{Hom}_{\mathbf{Set}}((A\times C)+(B\times C), X).\end{aligned}$$

By the appropriate application of Yoneda's lemma, we see that there is an isomorphism

$$(A+B)\times C \cong (A\times C) + (B\times C)$$

in **Fin**. The result about natural numbers follows. □

◊

5.2.1.22 The subobject classifier $\Omega \in \mathrm{Ob}(\mathcal{C}\text{–}\mathbf{Set})$

If $\mathcal{C}$ is a category then the functor category $\mathcal{C}$–**Set** is a very nice kind of category, called a *topos*. Note that when $\mathcal{C} = \underline{1}$ is the terminal category, then we have an isomorphism $\mathcal{C}$–**Set** $\cong$ **Set**, so the category of sets is a special case of a topos. What is so interesting about toposes (or topoi) is that they so nicely generalize many properties of **Set**. In this short section we investigate only one such property, namely that $\mathcal{C}$–**Set** has a subobject classifier, denoted $\Omega \in \mathrm{Ob}(\mathcal{C}\text{–}\mathbf{Set})$. In the case $\mathcal{C} = \underline{1}$, we saw back in Section 2.7.4.9 that the subobject classifier is $\{True, False\} \in \mathrm{Ob}(\mathbf{Set})$.

As usual, we consider the matter of subobject classifiers by grounding the discussion in terms of databases.

Definition 5.2.1.23. Let $\mathcal{C}$ be a category, let $\mathcal{C}$–**Set** denote its category of instances, and let $1 \in \mathrm{Ob}(\mathcal{C}\text{–}\mathbf{Set})$ denote the terminal object. A *subobject classifier for* $\mathcal{C}$–**Set** is an object $\Omega_{\mathcal{C}} \in \mathrm{Ob}(\mathcal{C}\text{–}\mathbf{Set})$ and a morphism $t\colon 1 \to \Omega_{\mathcal{C}}$ with the following property. For any monomorphism $f\colon X \to Y$ in $\mathcal{C}$–**Set**, there exists a unique morphism $char(f)\colon Y \to \Omega_{\mathcal{C}}$ such that the following diagram is a pullback in $\mathcal{C}$–**Set**:

$$\begin{array}{ccc} X & \xrightarrow{!} & 1 \\ {\scriptstyle f}\downarrow & \lrcorner & \downarrow{\scriptstyle t} \\ Y & \xrightarrow[char(f)]{} & \Omega_{\mathcal{C}} \end{array}$$

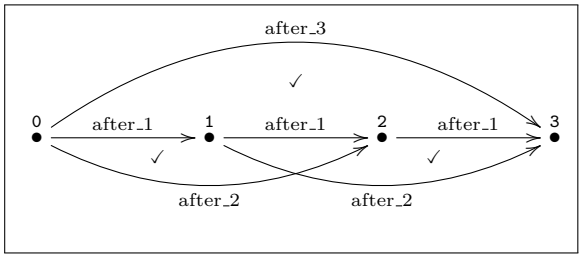
In terms of databases, what this means is that for every schema $\mathcal{C}$ there is some special instance $\Omega_{\mathcal{C}} \in \mathrm{Ob}(\mathcal{C}\text{–}\mathbf{Set})$ that somehow classifies sub-instances. When our schema is the terminal category, $\mathcal{C} = \underline{1}$, instances are sets and we saw in Definition 2.7.4.9 that the subobject classifier is $\Omega_{\underline{1}} = \{True, False\}$. One might think that the subobject classifier for $\mathcal{C}$–**Set** should just consist of a two-element set table-by-table, i.e. that for every $c \in \mathrm{Ob}(\mathcal{C})$ we should have $\Omega_{\mathcal{C}} =^{?} \{True, False\}$, but this is not correct.

In fact, for any object $c \in \mathrm{Ob}(\mathcal{C})$, it is easy to say what $\Omega_{\mathcal{C}}(c)$ should be. We know by Yoneda's lemma (Lemma 5.2.1.13) that $\Omega_{\mathcal{C}}(c) = \mathrm{Hom}_{\mathcal{C}\text{–}\mathbf{Set}}(Y_c, \Omega_{\mathcal{C}})$, where Y_c is the functor represented by c. There is a bijection between $\mathrm{Hom}_{\mathcal{C}\text{–}\mathbf{Set}}(Y_c, \Omega_{\mathcal{C}})$ and the set of sub-instances of Y_c. Each morphism $f\colon c \to d$ in $\mathcal{C}$ induces a morphism $Y_f\colon Y_d \to Y_c$, and the map $\Omega_{\mathcal{C}}(f)\colon \Omega_{\mathcal{C}}(c) \to \Omega_{\mathcal{C}}(d)$ sends a sub-instance $A \subseteq Y_c$ to the pullback

$$\begin{array}{ccc} Y_f^{-1}(A) & \longrightarrow & A \\ \downarrow & \lrcorner & \downarrow \\ Y_d & \xrightarrow[Y_f]{} & Y_c \end{array}$$

But this is all very abstract. We now give an example of a subobject classifier.

Example 5.2.1.24. Consider the category $\mathcal{C} \cong [3]$ depicted below

$\mathcal{C} :=$ 

To write down $\Omega_{\mathcal{C}}$ we need to understand the representable functors $Y_c \in \mathrm{Ob}(\mathcal{C}\text{–}\mathbf{Set})$, for $c = 0, 1, 2, 3$, as well as their subobjects. Here is Y_0 as an instance:

$Y_0(0)$

ID	**after_1**	**after_2**	**after_3**
☺	after_1(☺)	after_2(☺)	after_3(☺)

$Y_0(1)$

ID	**after_1**	**after_2**
after_1(☺)	after_2(☺)	after_3(☺)

$Y_0(2)$

ID	**after_1**
after_2(☺)	after_3(☺)

$Y_0(3)$

ID
after_3(☺)

What are the sub-instances of this? There is the empty sub-instance $\varnothing \subseteq Y_0$ and the identity sub-instance $Y_0 \subseteq Y_0$. But there are three more as well. Note that if we want to keep the ☺ row of table 0 then we have to keep everything. But if we throw away the ☺ row of table 0 we can still keep the rest and get a sub-instance. If we want to keep the after_1(☺) row of table 1 then we have to keep its images in tables 2 and 3. But we could throw away both the ☺ row of table 0 and the after_1(☺) row of table 1 and still keep the rest. And so on. In other words, the subobjects of Y_0 are in bijection with the set $\Omega_{\mathcal{C}}(0) := \{\textit{yes}, \textit{in 1}, \textit{in 2}, \textit{in 3}, \textit{never}\}$.

The same analysis holds for the other tables of $\Omega_{\mathcal{C}}$. It looks like this:

$\Omega_{\mathcal{C}}(0)$

ID	**after_1**	**after_2**	**after_3**
yes	*yes*	*yes*	*yes*
in 1	*yes*	*yes*	*yes*
in 2	*in 1*	*yes*	*yes*
in 3	*in 2*	*in 1*	*yes*
never	*never*	*never*	*never*

$\Omega_{\mathcal{C}}(1)$

ID	**after_1**	**after_2**
yes	*yes*	*yes*
in 1	*yes*	*yes*
in 2	*in 1*	*yes*
never	*never*	*never*

$\Omega_{\mathcal{C}}(2)$

ID	**after_1**
yes	*yes*
in 1	*yes*
never	*never*

$\Omega_{\mathcal{C}}(3)$

ID
yes
never

The morphism $1 \to \Omega_{\mathcal{C}}$ picks out the *yes* row of every table.

Now that we have constructed $\Omega_{\mathcal{C}} \in \mathrm{Ob}(\mathcal{C}\text{–}\mathbf{Set})$, we are ready to see it in action. What makes $\Omega_{\mathcal{C}}$ special is that for any instance $X\colon \mathcal{C} \to \mathbf{Set}$, the subinstances if X are in one-to-one correspondence with the morphisms $X \to \Omega_{\mathcal{C}}$. Consider the following arbitrary instance X, where the blue rows denote a sub-instance $A \subseteq X$.

$X(0)$

ID	**after 1**	**after 2**	**after 3**
a_1	b_1	c_1	d_1
a_2	b_2	c_1	d_1
a_3	b_2	c_1	d_1
a_4	b_3	c_2	d_2
a_5	b_5	c_3	d_1

$X(1)$

ID	**after 1**	**after 2**
b_1	c_1	d_1
b_2	c_1	d_1
b_3	c_2	d_2
b_4	c_1	d_1
b_5	c_3	d_1

$X(2)$

ID	**after 1**
c_1	d_1
c_2	d_2
c_3	d_1

$X(3)$

ID
d_1
d_2

(5.6)

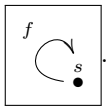
This blue sub-instance $A \subseteq X$ corresponds to a map $char(A)\colon X \to \Omega_{\mathcal{C}}$. That is for each $c \in \mathrm{Ob}(\mathcal{C})$ the rows in the c-table of X are sent to the rows in the c-table of $\Omega_{\mathcal{C}}$. The way $char(A)$ works is as follows. For each table i and row $x \in X(i)$, find the first column f in which the entry is blue (i.e. $f(x) \in A$), and send x to the corresponding element of $\Omega_{\mathcal{C}}(i)$. For example, $char(A)(\mathtt{0})$ sends a_1 to *in 2* and sends a_4 to *never*, and $char(A)(\mathtt{2})$ sends c_1 to *yes* and sends c_2 to *never*.

Exercise 5.2.1.25. a.) Write out the blue subinstance $A \subseteq X$ shown in (5.6) as an instance of $\mathcal{C}$, i.e. as four tables.

b.) This subinstance $A \subseteq X$ corresponds to a map $\ell := char(A)\colon X \to \Omega_{\mathcal{C}}$. For all $c \in \mathrm{Ob}(\mathcal{C})$ we have a function $\ell(c)\colon X(c) \to \Omega_{\mathcal{C}}(c)$. With $c = \mathtt{1}$, write out $\ell(\mathtt{1})\colon X(\mathtt{1}) \to \Omega_{\mathcal{C}}(\mathtt{1})$.

◊

Exercise 5.2.1.26. Let $\mathcal{L}oop$ be the loop schema

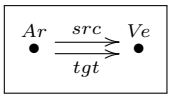
$\mathcal{L}oop =$ 

.

a.) What is the subobject classifier $\Omega_{\mathcal{L}oop} \in \mathrm{Ob}(\mathcal{L}oop\text{–}\mathbf{Set})$?

b.) How does $\Omega_{\mathcal{L}oop}$ compare to the representable functor Y_s?

◊

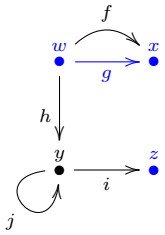
Exercise 5.2.1.27. Let $\mathbf{GrIn} =$ 

be the indexing category for graphs.

a.) Write down the subobject classifier $\Omega_{\mathbf{GrIn}} \in \mathrm{Ob}(\mathbf{GrIn}\text{–}\mathbf{Set})$ in tabular form, i.e. as two tables.

b.) Draw $\Omega_{\mathbf{GrIn}}$ as a graph.

c.) Let G be the graph below and $G' \subseteq G$ the blue part.



Write down $G \in \mathrm{Ob}(\mathbf{GrIn}\text{–}\mathbf{Set})$ in tabular form.

d.) Write down the components of the natural transformation $char(G')\colon G \to \Omega_{\mathbf{GrIn}}$.

◊

5.2.2 Database instances in other categories

5.2.2.1 Representations of groups

The classical mathematical subject of *representation theory* is the study of $\mathrm{Fun}(G, \mathbf{Vect})$ where G is a group and **Vect** is the category of vector spaces (over say $\mathbb{R}$). Every such

functor $F\colon G \to \mathbf{Vect}$ is called a *representation of* G. Since G is a category with one object $\blacktriangle$, F consists of a single vector space $V = F(\blacktriangle)$ together with an action of G on it.

We can think of this in terms of databases if we have a presentation of G in terms of generators and relations. The schema corresponding to G has one table and this table has a column for each generator. Giving a representation F is the same as giving an instance on our schema, with some properties that stem from the fact that our target category is **Vect** rather than **Set**. There are many possibilities for expressing [10] such data.

One possibility is if we could somehow draw V, say if V is 1-, 2-, or 3-dimensional. If so, let P be our chosen picture of V, e.g. P is the standard drawing of a Cartesian coordinate plane. Then every column of our table would consist entirely of the picture P instead of a set of rows. Drawing a point in the ID-column picture would result in a point being drawn in each other column's picture, in accordance with the G-action. Each column would of course respect addition and scalar multiplication.

Another possibility is to use the fact that there is a functor $U\colon \mathbf{Vect} \to \mathbf{Set}$, so our instance $F\colon G \to \mathbf{Vect}$ can be converted to an ordinary instance $U \circ F\colon G \to \mathbf{Set}$. We would have an ordinary set of rows. This set would generally be infinite, but it would be structured by addition and scalar multiplication. For example, assuming V is finite dimensional, one could find a few rows that generated the rest.

A third possibility is to use monads, which allow the table to have only as many rows as V has dimensions. This is a considerable savings of space. See Section 5.3.

5.2.2.2 Representations of quivers

Representation theory also studies representations of quivers. A *quiver* is just the free category (see Example 4.1.2.30) on a graph. If P is a graph with free category $\mathcal{P}$ then a representation of the quiver $\mathcal{P}$ is a functor $F\colon \mathcal{P} \to \mathbf{Vect}$. Such a representation consists of a vector space at every vertex of P and a linear transformation for every arrow. All of the discussion from Section 5.2.2.1 works in this setting, except that there is more than one table.

5.2.2.3 Other target categories

One can imagine the value of using target categories other than **Set** or **Vect** for databases.

Application 5.2.2.4. Geographic data consists of maps of the earth together with various functions on it. For example for any point on the earth one may want to know the average temperature recorded in the past 10 years, or the precise temperature at this moment. Earth can be considered as a topological space, E. Similarly, temperatures on earth reside on a continuum, say the space T of real numbers $[-100, 200]$. Thus the temperature record is a function $E \to T$.

Other records such as precipitation, population density, elevation, etc. can all be considered as continuous functions from E to some space. Agencies like the US Geological Survey hold databases of such information. By modeling them on functors $\mathcal{C} \to \mathbf{Top}$, they may be able to employ mathematical tools such as persistent homology [WeS] to find interesting invariants of the data.

◊◊

[10] We would use the term "representing" or "presenting", but they are both taken in the context of our narrative!

Application 5.2.2.5. Many other scientific disciplines could use the same kind of tool. For example, in studying the mechanics of materials, one may want to consider the material as a topological space M and measure values such as energy as a continuous $M \to E$. Such observations could be modeled by databases with target category **Top** or **Vect** rather than **Set**.

◊◊

5.2.3 Sheaves

Let X be a topological space (see Example 4.2.3.1), such as a sphere. In Section 5.2.2.3 we discussed continuous functions out of X, and their use in science (e.g. recording temperatures on the earth as a continuous map $X \to [-100, 200]$). Sheaves allow us to consider the local-global nature of such maps, taking into account reparable discrepancies in data gathering tools.

Application 5.2.3.1. Suppose that X is the topological space corresponding to the earth; by a *region* we mean an open subset $U \subseteq X$. Suppose that we cover X with 10,000 regions $U_1, U_2, \ldots, U_{10000}$, such that some of the regions overlap in a non-empty subregion (e.g. perhaps $U_5 \cap U_9 \neq \emptyset$). For each i, j let $U_{i,j} = U_i \cap U_j$.

For each region $U_i \subseteq X$ we have a temperature recording device, which gives a function $T_i \colon U_i \to [-100, 200]$. If $U_i \cap U_j \neq \emptyset$ then two different recording devices give us temperature data for the intersection $U_{i,j}$. Suppose we find that they do not give precisely the same data, but that there is a translation formula between their results. For example, T_i might register 3° warmer than T_j registers, throughout the region $U_i \cap U_j$.

A consistent system of translation formulas is called a *sheaf*. It does not demand a universal "true" temperature function, but only a consistent translation system between them.

◊◊

The following definitions (Definitions 5.2.3.2, 5.2.3.5) make the notion of sheaf precise, but we must go slowly (because it will already feel quick to the novice). For every region U, we can record the value of some function (say temperature) throughout U; although this record might consist of a mountain of data (a temperature for each point in U!), we think of it as one thing. That is, it is one element in the set of value-assignments throughout U. A sheaf holds the set of possible values-assignments-throughout-U's for all the different regions U, as well as how a value-assignment-throughout-U restricts to a value-assignment-throughout-V for any subset $V \subseteq U$.

Definition 5.2.3.2. Let X be a topological space, let $\mathrm{Open}(X)$ denote its partial order of open sets, and let $\mathrm{Open}(X)^{\mathrm{op}}$ be the opposite category. A *presheaf on X* is a functor $\mathcal{O} \colon \mathrm{Open}(X)^{\mathrm{op}} \to \mathbf{Set}$. For every open set $U \subseteq X$ we refer to the set $\mathcal{O}(U)$ as the *set of values-assignments throughout U of $\mathcal{O}$*. If $V \subseteq U$ is an open subset, it corresponds to an arrow in $\mathrm{Open}(X)$ and applying the functor $\mathcal{O}$ yields a function called the *restriction map from U to V* and denoted $\rho_{V,U} \colon \mathcal{O}(U) \to \mathcal{O}(V)$. Given $a \in \mathcal{O}(U)$, we may denote $\rho_{V,U}(a)$ by $a|_V$; it is called *the restriction of a to V*.

The *category of presheaves on X* is simply $\mathrm{Open}(X)^{\mathrm{op}}$–**Set**; see Definition 4.3.3.1.

Exercise 5.2.3.3.

a.) Come up with 4 overlapping open subsets that cover the square $X := [0, 3] \times [0, 3] \subseteq \mathbb{R}^2$. Write down a label for each open set as well as a label for each overlap (2-fold, 3-fold, etc.); you now have labeled n open sets. For each of these open sets, draw

a dot with the appropriate label, and then draw an arrow from one dot to another when the first refers to an open subset of the second. This is a preorder; call it $\mathrm{Open}(X)$. Now make up and write down formulas $R_1 \colon X \to \mathbb{R}$ and $R_2 \colon X \to \mathbb{R}$ with $R_1 \leqslant R_2$, expressing a range of temperatures $R_1(p) \leqslant x \leqslant R_2(p)$ that an imaginary experiment shows can exist at each point p in the square.

b.) Suppose we now tried to make our presheaf $\mathcal{O} \colon \mathrm{Open}(X)^{\mathrm{op}} \to \mathbf{Set}$ as follows. For each of your open sets, say A, we could put

$$\mathcal{O}(A) := \{f \colon A \to \mathbb{R} \mid R_1(a) \leqslant f(a) \leqslant R_2(a)\}.$$

What are the restriction maps? Do you like the name "value-assignment throughout A" for elements of $\mathcal{O}(A)$?

c.) We can now make another presheaf $\mathcal{O}'$ given the same experiment. For each of your open sets, say A, we could put

$$\mathcal{O}'(A) := \{f \colon A \to \mathbb{R} \mid f \text{ is continuous, and } R_1(a) \leqslant f(a) \leqslant R_2(a)\}.$$

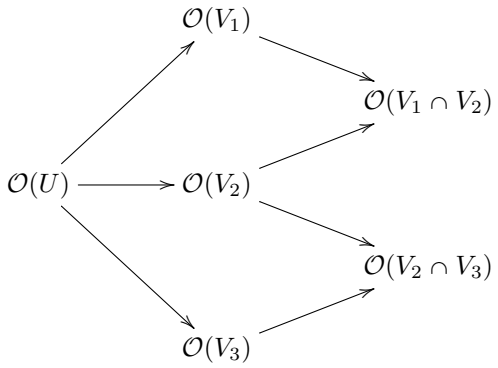
Are you comfortable with the idea that there is a morphism of presheaves $\mathcal{O}' \to \mathcal{O}$?

◊

Before we define sheaves, we need to clarify the notion of covering. Suppose that U is a region and that $V_1, \ldots, V_n$ are subregions (i.e. for each $1 \leqslant i \leqslant n$ we have $V_i \subseteq U$). Then we say that the V_i *cover* U if every point in U is in V_i for some i. Another way to say this is that the natural function $\sqcup_i V_i \to U$ is surjective.

Example 5.2.3.4. Let $X = \mathbb{R}$ be the space of real numbers, and define the following open subsets: $U = (5, 10), V_1 = (5, 7), V_2 = (6, 9), V_3 = (7, 10)$. [11] Then V_1, V_2, V_3 is a cover of U. It has overlaps $V_{12} = V_1 \cap V_2 = (6, 7)$, $V_{13} = V_1 \cap V_3 = \varnothing$, $V_{23} = V_2 \cap V_3 = (7, 9)$.

Given a presheaf $\mathcal{O} \colon \mathrm{Open}(X)^{\mathrm{op}} \to \mathbf{Set}$, we have sets and functions as in the following (incomplete) diagram



A presheaf $\mathcal{O}$ on X tells us what value-assignments throughout U can exist for each U. Suppose we have a value-assignment $a \in \mathcal{O}(U)$ throughout U and another value-assignment $a' \in \mathcal{O}(U')$ throughout U', and suppose that they agree as value-assignments throughout $U \cap U'$, i.e. $a|_{U \cap U'} = a'|_{U \cap U'}$. In this case we should have a unique value-assignment $b \in \mathcal{O}(U \cup U')$ throughout $U \cup U'$ that agrees on the U-part with a and agrees on the U'-part with a'; i.e. $b|_U = a$ and $b|_{U'} = a'$. This is the sheaf condition.

[11] We use parentheses to denote open intervals of real numbers. For example $(6, 9)$ denotes the set $\{x \in \mathbb{R} \mid 6 < x < 9\}$.

Definition 5.2.3.5. Let X be a topological space, let $\mathrm{Open}(X)$ be its partial order of open sets, and let $\mathcal{O}\colon \mathrm{Open}(X)^{\mathrm{op}} \to \mathbf{Set}$ be a presheaf. Given an open set $U \subseteq X$ and a cover $V_1, \ldots, V_n$ of U, the following condition is called the *sheaf condition* for that cover.

Sheaf condition Given a sequence $a_1, \ldots, a_n$ where each is a value-assignment $a_i \in \mathcal{O}(V_i)$ throughout V_i, suppose that for all i, j we have $a_i|_{V_i \cap V_j} = a_j|_{V_i \cap V_j}$; then there is a unique value-assignment $b \in \mathcal{O}(U)$ such that $b|_{V_i} = a_i$.

The presheaf $\mathcal{O}$ is called a *sheaf* if it satisfies the sheaf condition for every cover.

Example 5.2.3.6. Let $X = \mathbb{R}$ and let U, V_1, V_2, V_3 be the open cover given in Example 5.2.3.4. Given a measurement taken throughout V_1, a measurement taken throughout V_2, and a measurement taken throughout V_3, we have elements $a_1 \in \mathcal{O}(V_1), a_2 \in \mathcal{O}(V_2)$, and $a_3 \in \mathcal{O}(V_3)$. If they are in agreement on the overlap intervals, we can *glue* them to give a measurement throughout U.

Remark 5.2.3.7. In Application 5.2.3.1, we said that sheaves would help us patch together information from different sources. Even if different temperature-recording devices T_i and T_j registered different temperatures on an overlapping region $U_i \cap U_j$, we said they could be patched together if there was a consistent translation system between their results. What is actually needed is a set of isomorphisms

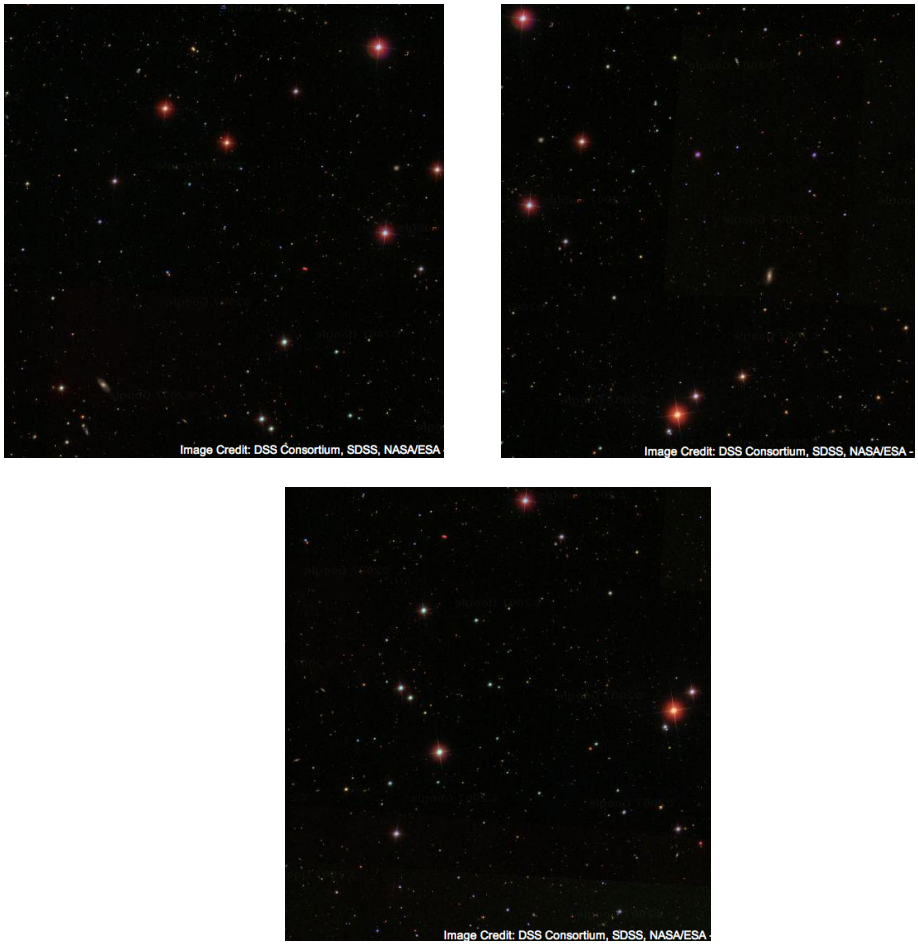
$$p_{i,j}\colon T_i|_{U_{i,j}} \xrightarrow{\cong} T_j|_{U_{i,j}}$$

that translate between them, and that these $p_{i,j}$'s act in concert with one another. This (when precisely defined,) is called *descent data*.. The way it interacts with our definition of sheaf given in Definitions 5.2.3.2 and 5.2.3.5 is buried in the restriction maps ρ for the overlaps as subsets $U_{i,j} \subseteq U_i$ and $U_{i,j} \subseteq U_j$. We will not explain further here. One can see [Gro].

Application 5.2.3.8. Consider outer space as a topological space X. Different astronomers record observations. Let $C = [390, 700]$ denote the set of wavelengths in the visible light spectrum (written in nanometers). Given an open subset $U \subseteq X$ let $\mathcal{O}(U)$ denote the set of functions $U \to C$. The presheaf $\mathcal{O}$ satisfies the sheaf condition; this is the taken-for-granted fact that we can patch together different observations of space.

Below are three views of the night sky. Given a telescope position to obtain the first view, one moves the telescope right and a little down to obtain the second and one moves it down and left to obtain the third. [12]

[12] Image credit: NASA, ESA, Digitized Sky Survey Consortium.



These are value-assignments $a_1 \in \mathcal{O}(V_1), a_2 \in \mathcal{O}(V_2)$, and $a_3 \in \mathcal{O}(V_3)$ throughout subsets $V_1, V_2, V_3 \subseteq X$ (respectively). These subsets V_1, V_2, V_3 cover some (strangely-shaped) subset $U \subseteq X$. The sheaf condition says that these three value-assignments glue together to form a single value-assignment throughout U:

◊◊

Exercise 5.2.3.9. Find an application of sheaves in your own domain of expertise. ◊

Application 5.2.3.10. Suppose we have a sheaf for temperatures on earth. For every region U we have a set of theoretically possible temperature-assignments throughout U. For example we may know that if it is warm in Texas, warm in Arkansas, and warm in Kansas, then it cannot be cold in Oklahoma. With such a sheaf $\mathcal{O}$ in hand, one can use facts about the temperature in one region U to predict the temperature in another region V.

The mathematics is as follows. Suppose given regions $U, V \subseteq X$ and a subset $A \subseteq \mathcal{O}(U)$ corresponding to what we know about the temperature assignment throughout U. We take the following fiber product

$$\begin{array}{ccccc} (\rho_{U,X})^{-1}(A) & \longrightarrow & \mathcal{O}(X) & \xrightarrow{\rho_{V,X}} & \mathcal{O}(V) \\ \downarrow & \lrcorner & \downarrow{\scriptstyle \rho_{U,X}} & & \\ A & \longrightarrow & \mathcal{O}(U) & & \end{array}$$

The image of the top map is a subset of $\mathcal{O}(V)$ telling us which temperature-assignments are possible throughout V given our knowledge A about the temperature throughout U.

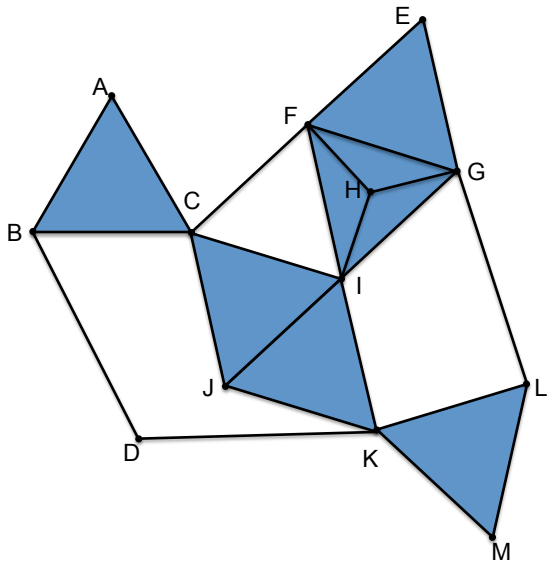
We can imagine the same type of prediction systems for other domains as well, such as the energy of various parts of a material. ◊◊

Example 5.2.3.11. In Exercises 4.2.4.3 and 4.2.4.4 we discussed the idea of laws being dictated or respected throughout a jurisdiction. If X is earth, to every jurisdiction $U \subseteq X$ we assign the set $\mathcal{O}(U)$ of laws that are dictated to hold throughout U. Given a law on U and a law on V, we can see if they amount to the same law on $U \cap V$. For example, on U a law might say "no hunting near rivers" and on V a law might say "no hunting in public areas". It just so happens that on $U \cap V$ all public areas are near rivers and vice versa, so the laws agree there. These laws patch together to form a single rule about hunting that is enforced throughout the union $U \cup V$, respected by all jurisdictions within it.

5.2.3.12 Sheaf of ologged concepts

Definition 5.2.3.5 defines what should be called a sheaf of sets. We can discuss sheaves of groups or even sheaves of categories. Here is an application of the latter.

Recall the notion of simplicial complexes discussed in Section 2.7.4.3. They look like this:



(5.7)

Given such a simplicial complex X, we can imagine each vertex $v \in X_0$ as an entity with a worldview (e.g. a person) and each simplex as the common worldview shared by its vertices. To model this, we will assign to each vertex $v \in X$ an olog $\mathcal{O}(v)$, corresponding to the worldview held by that entity, and to each simplex $u \in X_n$, we assign an olog $\mathcal{O}(u)$ corresponding to a *common ground* worldview.. Recall that X is a subset of $\mathbb{P}(X_0)$; it is a preorder and its elements (the simplices) are ordered by inclusion. If u, v are simplices with $u \subseteq v$ then we want a map of ologs (i.e. a schema morphism) $\mathcal{O}(v) \to \mathcal{O}(u)$ corresponding to how any idea that is shared among the people in v is shared among the people in u. Thus we have a functor $\mathcal{O}\colon X \to \mathbf{Sch}$ (where we are forgetting the distinction between ologs and databases for notational convenience).

To every simplicial complex (indeed every ordered set) one can associate a topological space; in fact we have a functor $Alx\colon \mathbf{PrO} \to \mathbf{Top}$, called the Alexandrov functor. Applying $Alx(X^{\mathrm{op}})$ we have a space which we denote by $\mathcal{X}$. One can visualize $\mathcal{X}$ as X, but the open sets include unions of simplices. There is a unique sheaf of categories on $\mathcal{X}$ that behaves like X on simplices.

How does this work in the case of our sheaf $\mathcal{O}$ of worldviews? For simplices such as (A) or (CI), the sheaf returns the olog corresponding to that person or shared worldview. But for open sets like the union of (CIJ) and (IJK), what we get is the olog consisting of the types shared by C, I, and J for which I and J affirm agreement with types shared by I, J, and K.

Example 5.2.3.13. Imagine two groups of people G_1 and G_2 each making observations about the world. Suppose that there is some overlap $H = G_1 \cap G_2$. Then it may happen that there is a conversation including G_1 and G_2 and both groups are talking about something and, although using different words, H says "you guys are talking about the

same things, you just use different words." In this case there is an object-assignment throughout $G_1 \cup G_2$ that agrees with both those on G_1 and those on G_2.

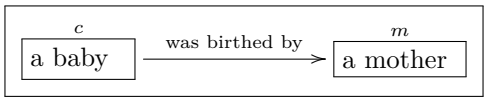
5.2.3.14 Time

One can use sheaves to model objects in time; Goguen gave an approach to this in [Gog]. For another approach, let $\mathcal{C}$ be a database schema. The lifespan of information about the world is generally finite; that is, what was true yesterday is not always the case today. Thus we can associate to each interval U of time the information that we deem to hold throughout U. This is sometimes called the *valid time* of the data.

If something is the case throughout U and we have a subset $V \subseteq U$ then of course it is the case throughout V. And the sheaf condition holds too: if some information holds throughout U and some other information holds throughout U', and if these two things restrict to the same information on the overlap $U \cap V$, then they can be glued to information that holds throughout the union $U \cup V$.

So we can model information-change over time by using a sheaf of $\mathcal{C}$-sets on the topological space $\mathbb{R}$. One way to think of this is simply as an instance on the schema $\mathcal{C} \times \mathrm{Open}(\mathbb{R})^{\mathrm{op}}$. The sheaf condition is just an added property that our instances have to obey.

Example 5.2.3.15. Consider a hospital in which babies are born. In our scenario, mothers enter the hospital, babies are born, mothers and babies leave the hospital. Let $\mathcal{C}$ be the schema



Consider the 8-hour intervals

$$\begin{aligned}
\mathrm{Shift}_1 &:= (\text{Jan } 1 - 00:00,\ \text{Jan } 1 - 08:00),\\
\mathrm{Shift}_2 &:= (\text{Jan } 1 - 04:00,\ \text{Jan } 1 - 12:00),\\
\mathrm{Shift}_3 &:= (\text{Jan } 1 - 8:00,\ \text{Jan } 1 - 16:00).
\end{aligned}$$

The nurses take shifts of 8 hours, overlapping with their predecessors by 4 hours, and they record in the database only patients that were there throughout their shift or throughout any overlapping shift. A mother might be in the hospital throughout shift 1, arriving before the new year. A baby is born at 05:00 on Jan 1, and thus does not make it into the Shift_1-table, but does make it into the $(\mathrm{Shift}_1 \cap \mathrm{Shift}_2)$-table. The two are there until 17:00 on Jan 1, and so they are recorded in the Shift_2 and Shift_3 tables.

Whether or not this implementation of the sheaf semantics is most useful in practice is certainly debatable. But something like this could easily be useful as a semantics, i.e. a way of thinking about, the temporal nature of data.

5.3 Monads

Monads would probably not have been invented without category theory, but they have been quite useful in formalizing algebra, calculating invariants of topological spaces, and imbedding non-functional operations into functional programming languages. We will mainly discuss monads in terms of how they can help us make modeling contexts explicit, and in so doing allow us to simplify the language we use in the model.

Much of the following material on monads is taken from [Sp3].

5.3.1 Monads formalize context

Monads can formalize assumptions about the way one will do business throughout a domain. For example, suppose that we want to consider functions that do not have to return a value for all inputs. Such *partial functions* can be composed. Indeed, given a partial function $f\colon A \to B$ and a partial function $g\colon B \to C$, one gets a partial function $g \circ f\colon A \to C$ in an obvious way.

Here we are drawing arrows as though we are talking about functions, but there is an implicit context in which we are actually talking about partial functions. Monads allow us to write things in the "functional" way while holding the underlying context. What makes them useful is that the notion of *context* we are using here is made formal.

Example 5.3.1.1 (Partial functions). Partial functions can be modeled by ordinary functions, if we add a special "no answer" element to the codomain. That is, the set of partial functions $A \to B$ is in one-to-one correspondence with the set of ordinary functions $A \to B \sqcup \{\text{☹}\}$. For example, suppose we want to model the partial function $f(x) := \frac{1}{x^2-1}\colon \mathbb{R} \to \mathbb{R}$ in this way, we would use the function

$$f(x) := \begin{cases} \frac{1}{x^2-1} & \text{if } x \neq -1 \text{ and } x \neq 1, \\ \text{☹} & \text{if } x = -1, \\ \text{☹} & \text{if } x = 1. \end{cases}$$

An ordinary function $f\colon A \to B$ can be considered a partial function because we can compose with the inclusion

$$B \to B \sqcup \{\text{☹}\} \tag{5.8}$$

But how do we compose two partial functions written in this way? Suppose $f\colon A \to B \sqcup \{\text{☹}\}$ and $g\colon B \to C \sqcup \{\text{☹}\}$ are functions. First form a new function

$$g' := g \sqcup \{\text{☹}\}\colon B \sqcup \{\text{☹}\} \to C \sqcup \{\text{☹}\} \sqcup \{\text{☹}\} \tag{5.9}$$

then compose to get $(g' \circ f)\colon A \to C \sqcup \{\text{☹}\} \sqcup \{\text{☹}\}$, and finally send both ☹'s to the same element by composing with

$$C \sqcup \{\text{☹}\} \sqcup \{\text{☹}\} \to C \sqcup \{\text{☹}\}. \tag{5.10}$$

What does this mean? Every element $a \in A$ is sent by f to either an element $b \in B$ or "no answer". If it has an answer $f(a) \in B$, this is either sent by g to an element $g(f(a)) \in C$ or to "no answer". We get a partial function $A \to C$ by sending a to $g(f(a))$ if possible or to "no answer" if it gets stopped along the way.

This monad is sometimes called the *maybe monad* in computer science, because a partial function $f\colon A \to B$ takes every element of A and either outputs just an element of B or outputs nothing; more succinctly, it outputs a "maybe B".

Application 5.3.1.2. Experiments are supposed to be performed objectively, but suppose we imagine that changing the person who performs the experiment, say in psychology, may change the outcome. Let A be the set of experimenters, let X be the parameter space for the experimental variables (e.g. $X = \text{Age} \times \text{Income}$) and let Y be the observation space (e.g. $Y =$ propensity for violence). Then whereas we want to think of such an experiment as telling us about a function $f\colon X \to Y$, we may want to make some of the context explicit by including information about who performed the experiment. That is, we are really finding a function $f\colon X \times A \to Y$.

However, it may be the case that even ascertaining someones age or income, which is done by asking that person, is subject to who in A is doing the asking, and so we again want to consider the experimenter as part of the equation. In this case, we can use a monad to hide the fact that everything in sight is assumed to be influenced by A. In other words, we want to announce once and for all our modeling context—that every observable is possibly influenced by the observer—so that it can recede into the background.

We will return to this in Examples 5.3.2.6 and 5.3.3.4.

◊◊

5.3.2 Definition and examples

What aspects of Example 5.3.1.1 are really about monads, and what aspects are just about partial functions in particular? It is a functor and a pair of natural transformations that showed up in (5.9), (5.8), and (5.10). In this section we will give the definition and a few examples. We will return to our story about how monads formalize context in Section 5.3.3.

Definition 5.3.2.1 (Monad)**.** A *monad on* **Set** is defined as follows: One announces some constituents (A. functor, B. unit map, C. multiplication map) and asserts that they conform to some laws (1. unit laws, 2. associativity law). Specifically, one announces

A. a functor $T\colon \mathbf{Set} \to \mathbf{Set}$,

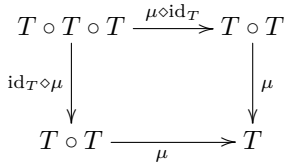
B. a natural transformation $\eta\colon \mathrm{id}_{\mathbf{Set}} \to T$, and

C. a natural transformation $\mu\colon T \circ T \to T$

We sometimes refer to the functor T as though it were the whole monad; we call η the *unit map* and we call μ the *multiplication map*. One asserts that the following laws hold:

1. The following diagrams of functors **Set** → **Set** commute:

2. The following diagram of functors **Set** → **Set** commutes:



Example 5.3.2.2 (List monad). We now go through Definition 5.3.2.1 using what is called the List monad. The first step is to give a functor $\mathrm{List}\colon \mathbf{Set} \to \mathbf{Set}$, which we did in Example 4.1.2.18. Recall that if $X = \{p, q, r\}$ then $\mathrm{List}(X)$ includes the empty list $[\,]$, singleton lists, such as $[p]$, and any other list of elements in X, such as $[p, p, r, q, p]$. Given

a function $f\colon X \to Y$, one obtains a function $\mathrm{List}(f)\colon \mathrm{List}(X) \to \mathrm{List}(Y)$ by entry-wise application of f.

As a monad, the functor List comes with two natural transformations, a unit map η and a multiplication map μ. Given a set X, the unit map $\eta_X\colon X \to \mathrm{List}(X)$ returns singleton lists as follows

$$\begin{array}{ccc} X & \xrightarrow{\eta_X} & \mathrm{List}(X) \\ \\ p & \longmapsto & [p] \\ q & \longmapsto & [q] \\ r & \longmapsto & [r] \end{array}$$

Given a set X, the multiplication map $\mu_X\colon \mathrm{List}(\mathrm{List}(X)) \to \mathrm{List}(X)$ flattens lists of lists as follows.

$$\begin{array}{ccc} \mathrm{List}(\mathrm{List}(X)) & \xrightarrow{\mu_X} & \mathrm{List}(X) \\ \\ [[q,p,r],[],[q,r,p,r],[r]] & \longmapsto & [q,p,r,q,r,p,r,r] \end{array}$$

The naturality of η and μ just mean that these maps work appropriately well under term-by-term replacement by a function $f\colon X \to Y$. Finally the three monad laws from Definition 5.3.2.1 can be exemplified as follows:

$$\begin{array}{ccc} [p,q,q] & \xmapsto{\mathrm{id}_{\mathrm{List}} \circ \eta} & [[p],[q],[q]] \\ & \searrow & \downarrow{\scriptstyle \mu} \\ & & [p,q,q] \end{array} \qquad \begin{array}{ccc} [p,q,q] & \xmapsto{\eta \circ \mathrm{id}_{\mathrm{List}}} & [[p,q,q]] \\ & \searrow & \downarrow{\scriptstyle \mu} \\ & & [p,q,q] \end{array}$$

$$\begin{array}{ccc} \big[[[p,q],[r]],[[],[r,q,q]]\big] & \xmapsto{\mu \circ \mathrm{id}_{\mathrm{List}}} & [[p,q],[r],[],[r,q,q]] \\ \downarrow{\scriptstyle \mathrm{id}_{\mathrm{List}} \circ \mu} & & \downarrow{\scriptstyle \mu} \\ [[p,q,r],[r,q,q]] & \xmapsto[\mu]{} & [p,q,r,r,q,q] \end{array}$$

Exercise 5.3.2.3. Let $\mathbb{P}\colon \mathbf{Set} \to \mathbf{Set}$ be the powerset functor, so that given a function $f\colon X \to Y$ the function $\mathbb{P}(f)\colon \mathbb{P}(X) \to \mathbb{P}(Y)$ is given by taking images.

a.) Make sense of the following statement: "with η defined by singleton subsets and with μ defined by union, $\top := (\mathbb{P}, \eta, \mu)$ is a monad".

b.) With $X = \{a, b\}$, write down the function η_X as a 2-row, 2-column table, and write down the function μ_X as a 16-row, 2-column table (you can stop after 5 rows if you fully get it).

c.) Check that you believe the monad laws from Definition 5.3.2.1.

◊

Example 5.3.2.4 (Partial functions as a monad). Here is the monad for partial functions. The functor $T\colon \mathbf{Set} \to \mathbf{Set}$ sends a set X to the set $X \sqcup \{\smiley\}$. Clearly, given a function $f\colon X \to Y$ there is an induced function $f \sqcup \{\smiley\}\colon X \sqcup \{\smiley\} \to Y \sqcup \{\smiley\}$, so this is a functor. The natural transformation $\eta\colon \mathrm{id} \to T$ is given on a set X by the component function

$$\eta_X\colon X \to X \sqcup \{\smiley\}$$

that includes $X \hookrightarrow X \sqcup \{\smiley\}$. Finally, the natural transformation $\mu\colon T \circ T \to T$ is given on a set X by the component function

$$\mu_X\colon X \sqcup \{\smiley\} \sqcup \{\smiley\} \longrightarrow X \sqcup \{\smiley\}$$

that collapses both copies of $\smiley$.

Exercise 5.3.2.5. Let E be a set, elements we will refer to as *exceptions*. We imagine that a function $f\colon X \to Y$ either outputs a value or one of these exceptions, which might be things like "overflow!" or "division by zero!", etc. Let $T\colon \mathbf{Set} \to \mathbf{Set}$ be the functor $X \mapsto X \sqcup E$. Follow Example 5.3.2.4 and come up with a unit map η and a multiplication map μ for which (T, η, μ) is a monad. ◊

Example 5.3.2.6. Fix a set A. Let $T\colon \mathbf{Set} \to \mathbf{Set}$ be given by $T(X) = X^A = \mathrm{Hom}_{\mathbf{Set}}(A, X)$; this is a functor. For a set X, let $\eta_X\colon X \to T(X)$ be given by the constant function, $x \mapsto c_x\colon A \to X$ where $c_x(a) = x$ for all $a \in A$. To specify a function

$$\mu_X\colon \mathrm{Hom}_{\mathbf{Set}}(A, T(X)) \to \mathrm{Hom}_{\mathbf{Set}}(A, X),$$

we curry and need a function $A \times \mathrm{Hom}_{\mathbf{Set}}(A, T(X)) \to X$. We have an evaluation function (see Exercise 2.7.2.5) $ev\colon A \times \mathrm{Hom}_{\mathbf{Set}}(A, T(X)) \to T(X)$, and we have an identity function $\mathrm{id}_A\colon A \to A$, so we have a function $(\mathrm{id}_A \times ev)\colon A \times \mathrm{Hom}_{\mathbf{Set}}(A, T(X)) \longrightarrow A \times T(X)$. Composing that with another evaluation function $A \times \mathrm{Hom}_{\mathbf{Set}}(A, X) \to X$ yields our desired μ_X. Namely, for all $b \in A$ and $f \in \mathrm{Hom}(A, T(X))$ we have

$$\mu_X(f)(b) = f(b)(b).$$

Remark 5.3.2.7. Monads can be defined on categories other than **Set**. In fact, for any category $\mathcal{C}$ one can take Definition 5.3.2.1 and replace every occurrence of **Set** with $\mathcal{C}$ and obtain the definition for monads on $\mathcal{C}$. We have actually seen a monad $(\mathrm{Paths}, \eta, \mu)$ on the category **Grph** of graphs before, namely in Examples 4.3.1.12 and 4.3.1.13. That is, Paths: **Grph** → **Grph**, which sends a graph to its paths-graph is the functor part. The unit map η includes a graph into its paths-graph using the observation that every arrow is a path of length 1. And the multiplication map μ concatenates paths of paths. The Kleisli category of this monad (see Definition 5.3.3.1) is used, e.g. in (4.14) to define morphisms of database schemas.

5.3.3 Kleisli category of a monad

Given a monad $\top := (T, \eta, \mu)$, we can form a new category $\mathbf{Kls}(\top)$.

Definition 5.3.3.1. Let $\top = (T, \eta, \mu)$ be a monad on **Set**. Form a new category, called the *Kleisli category for* $\top$, denoted $\mathbf{Kls}(\top)$, with sets as objects, $\mathrm{Ob}(\mathbf{Kls}(\top)) := \mathrm{Ob}(\mathbf{Set})$, and with

$$\mathrm{Hom}_{\mathbf{Kls}(\top)}(X, Y) := \mathrm{Hom}_{\mathbf{Set}}(X, T(Y))$$

for sets X, Y. The identity morphism $\mathrm{id}_X : X \to X$ in $\mathbf{Kls}(\top)$ is given by $\eta : X \to T(X)$ in $\mathbf{Set}$. The composition of morphisms $f : X \to Y$ and $g : Y \to Z$ in $\mathbf{Kls}(\top)$ is given as follows. Writing them as functions, we have $f : X \to T(Y)$ and $g : Y \to T(Z)$. The first step is to apply the functor T to g, giving $T(g) : T(Y) \to T(T(Z))$. Then compose with f to get $T(g) \circ f : X \to T(T(Z))$. Finally, compose with $\mu_Z : T(T(Z)) \to T(Z)$ to get the required function $X \to T(Z)$. The associativity of this composition formula follows from the associativity law for monads.

Example 5.3.3.2. Recall the monad $\top$ for partial functions, $T(X) = X \sqcup \{\text{☺}\}$, from Example 5.3.2.4. The Kleisli category $\mathbf{Kls}(\top)$ has sets as objects, but a morphism $f : X \to Y$ means a function $X \to Y \sqcup \{\text{☺}\}$, i.e a partial function. Given another morphism $g : Y \to Z$, the composition formula in $\mathbf{Kls}(\top)$ ensures that $g \circ f : X \to Z$ has the appropriate behavior.

Note how this monad allows us to make explicit our assumption that all functions are partial, and then hide it away from our notation.

Remark 5.3.3.3. For any monad $\top = (T, \eta, \mu)$ on $\mathbf{Set}$, there is a functor $i : \mathbf{Set} \to \mathbf{Kls}(\top)$ given as follows. On objects we have $\mathrm{Ob}(\mathbf{Kls}(\top)) = \mathrm{Ob}(\mathbf{Set})$, so take $i = \mathrm{id}_{\mathrm{Ob}(\mathbf{Set})}$. Given a morphism $f : X \to Y$ in $\mathbf{Set}$, we need a morphism $i(f) : X \to Y$ in $\mathbf{Kls}(\top)$, i.e. a function $i(f) : X \to T(Y)$. We assign $i(f)$ to be the composite $X \xrightarrow{f} Y \xrightarrow{\eta} T(Y)$. The functoriality of this mapping follows from the unit law for monads.

The point is that any ordinary function (morphism in $\mathbf{Set}$) has an interpretation as a morphism in the Kleisli category of any monad. More categorically, there is a functor $\mathbf{Set} \to \mathbf{Kls}(\top)$.

Example 5.3.3.4. In this example we return to the setting laid out by Application 5.3.1.2 where we had a set A of experimenters and assumed that the person doing the experiment may affect the outcome. We use the monad $\top = (T, \eta, \mu)$ from Example 5.3.2.6 and hope that $\mathbf{Kls}(\top)$ will conform to our understanding of how to manage the affect of the experimenter on data.

The objects of $\mathbf{Kls}(\top)$ are ordinary sets, but a map $f : X \to Y$ in $\mathbf{Kls}(\top)$ is a function $X \to Y^A$. By currying this is the same as a function $X \times A \to Y$, as desired. To compose f with $g : Y \to Z$ in $\mathbf{Kls}(\top)$, we follow the formula. It turns out to be equivalent to the following. We have a function $X \times A \to Y$ and a function $Y \times A \to Z$. Modifying the first slightly, we have a function $X \times A \to Y \times A$, by identity on A, and we can now compose to get $X \times A \to Z$.

What does this say in terms of experimenters affecting data gathering? It says that if we work within $\mathbf{Kls}(\top)$ then we will be able to assume that the experimenter is being taken into account; all proposed functions $X \to Y$ are actually functions $A \times X \to Y$. The natural way to compose these experiments is that we only consider the data from one experiment to feed into another if the experimenter is the same in both experiments. [13]

Exercise 5.3.3.5. In Exercise 5.3.2.3 we discussed the power set monad $\top = (\mathbb{P}, \eta, \mu)$.

a.) Can you find a way to relate the morphisms in $\mathbf{Kls}(\top)$ to relations? That is, given a morphism $f : A \to B$ in $\mathbf{Kls}(\top)$, is there a natural way to associate to it a relation $R \subseteq A \times B$?

[13] This requirement seems a bit stringent, but it can be mitigated in a variety of ways. One such way is to notice that by Remark 5.3.3.3 that we have not added any requirement, because any old way of doing business yields a valid new way of doing business (we just say "every experimenter would get the same result"). Another way would be to hand off the experiment results to another person, who could carry it forward (see Example 5.3.3.8).

b.) How does the composition formula in $\mathbf{Kls}(\top)$ relate to the composition of relations given in Definition 2.5.2.3? [14]

◊

Exercise 5.3.3.6. Let $\top = (\mathbb{P}, \eta, \mu)$ be the power set monad. The category $\mathbf{Kls}(\top)$ is closed under binary products, i.e. every pair of objects $A, B \in \mathrm{Ob}(\mathbf{Kls}(\top))$ have a product in $\mathbf{Kls}(\top)$. What is the product of $A = \{1, 2, 3\}$ and $B = \{a, b\}$? ◊

Exercise 5.3.3.7. Let $\top = (\mathbb{P}, \eta, \mu)$ be the power set monad. The category $\mathbf{Kls}(\top)$ is closed under binary coproducts, i.e. every pair of objects $A, B \in \mathrm{Ob}(\mathbf{Kls}(\top))$ have a coproduct in $\mathbf{Kls}(\top)$. What is the coproduct of $A = \{1, 2, 3\}$ and $B = \{a, b\}$? ◊

Example 5.3.3.8. Let A be any preorder. We speak of A throughout this example as though it was the linear order given by time because this is a nice case, however the mathematics works for any $A \in \mathrm{Ob}(\mathbf{PrO})$.

There is a monad $\top = (T, \eta, \mu)$ that captures the idea that a function $f\colon X \to Y$ occurs in the context of time in the following sense: The output of f is determined not only by the element $x \in X$ on which it is applied but also by the time at which it was applied to x; and the output of f occurs at another time, which is not before the time of input.

The functor part of the monad is given on $X \in \mathrm{Ob}(\mathbf{Set})$ by

$$T(X) = \{p\colon A \to A \times X \mid \text{ if } p(a) = (a', x) \text{ then } a' \geqslant a\}.$$

The unit $\eta_X\colon X \to T(X)$ sends x to the function $a \mapsto (a, x)$. The multiplication map $\mu_X\colon T(T(X)) \to T(X)$ is roughly described as follows. If for every $a \in A$ you have a later element $a' \geqslant a$ and a function $p\colon A \to A \times X$ that takes elements of A to later elements of A and values of X, then $p(a')$ is a still later element of A and a value of X, as desired.

Morphisms in the Kleisli category $\mathbf{Kls}(\top)$ can be curried to be functions $f\colon A \times X \to A \times Y$ such that if $f(a, x) = (a', y)$ then $a' \geqslant a$.

Remark 5.3.3.9. One of the most important monads in computer science is the so-called *state monad*. It is used when one wants to allow a program to mutate state variables (e.g. in the program

if $x > 4$ then $x := x + 1$ else Print "done")

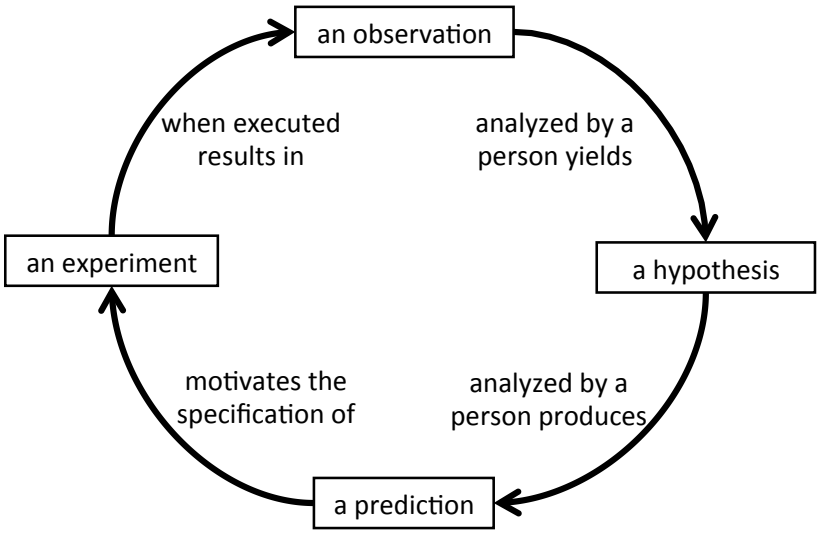
x is a state variable. The state monad is a special case of the monad discussed in Example 5.3.3.8. Given any set A, the usual *state monad of type* A is obtained by giving A the indiscrete preorder (see Example 3.4.4.5). More explicitly it is a monad with functor part

$$X \mapsto (A \times X)^X,$$

and it will be briefly discussed in Example 5.3.5.4.

Example 5.3.3.10. Here we reconsider the image from the front cover of this book, reproduced here.

[14] Actually, Definition 2.5.2.3 is about composing spans, but a relation $R \subseteq A \times B$ is a kind of span, $R \to A \times B$.



It looks like an olog, and all ologs are database schemas (see Section 3.5.2.14). But how is "analyzed by a person yields" a function from observations to hypotheses? The very name belies the fact that it is an invalid aspect in the sense of Section 2.3.2.1, because given an observation there may be more than one hypothesis yielded, corresponding to which person is doing the observing. In fact, all of the arrows in this diagram correspond to some hidden context involving people: the prediction is dependent on who analyzes the hypothesis, the specification of an experiment is dependent on who is motivated to specify it, and experiments may result in different observations by different observers.

Without monads, the model of science proposed by this olog would be difficult to believe in. But by choosing a monad we can make explicit (and then hide from discourse) our implicit assumption that "of course this is all dependent on which human is doing the science". The choice of monad is an additional modeling choice. Do we want to incorporate the partial order of time? Do we want the scientist to be modified by each function (i.e. the person is changed when analyzing an observation to yield a hypothesis)? These are all interesting possibilities.

One reasonable choice would be to use the state monad of type A, where A is the set of scientific models. This implies the following context: every morphism $f\colon X \to Y$ in the Kleisli category of this monad is really a morphism $f\colon X \times A \to Y \times A$; while ostensibly giving a map from X to Y, it is influenced by the scientific model under which it is performed, and its outcome yields a new scientific model.

Reading the olog in this context might look like this:

> A hypothesis (in the presence of a scientific model) analyzed by a person produces a prediction (in the presence of a scientific model), which motivates the specification of an experiment (in the presence of a scientific model), which when executed results in an observation (in the presence of a scientific model), which analyzed by a person yields a hypothesis (in the presence of a scientific model).

The parenthetical statements can be removed if we assume them to always be around, which can be done using the monad above.

5.3.3.11 Relaxing functionality constraint for ologs

In Section 2.3.2 we said that every arrow in an olog has to be English-readable as a sentence, and it has to correspond to a function. For example, the arrow

$$\boxed{\text{a person}} \xrightarrow{\text{has}} \boxed{\text{a child}} \tag{5.11}$$

comprises an readable sentence, but does not correspond to a function because a person may have no children or more than one child. We'll call olog in which every arrow corresponds to a function (the only option proposed so far in the book) a *functional olog*. Requiring that ologs be functional as we have been doing, comes with advantages and disadvantages. The main advantage is that creating a functional olog requires more conceptual clarity about the situation, and this has benefits for the olog-creator as well as for anyone to whom he or she tries to explain the situation. The main disadvantage is that creating a functional olog takes more time, and the olog takes up more space on the page.

In the context of the power set monad (see Exercise 5.3.2.3), a morphism $f\colon X \to Y$ between sets X and Y becomes a binary relation on X and Y, rather than a function, as seen in Exercise 5.3.3.5. So in that context, the arrow in (5.11) becomes valid. An olog in which arrows correspond to mere binary relations rather than functions might be called a *relational olog*.

5.3.4 Monads in databases

In this section we discuss how to record data in the presence of a monad. The idea is quite simple. Given a schema (category) $\mathcal{C}$, an ordinary instance is a functor $I\colon \mathcal{C} \to \mathbf{Set}$. But if $\top = (T, \eta, \mu)$ is a monad, then a *Kleisli* $\top$*-instance on* $\mathcal{C}$ is a functor $J\colon \mathcal{C} \to \mathbf{Kls}(\top)$. Such a functor associates to every object $c \in \mathrm{Ob}(\mathcal{C})$ a set $J(c)$, and to every arrow $f\colon c \to c'$ in $\mathcal{C}$ a morphism $J(f)\colon J(c) \to J(c')$ in $\mathbf{Kls}(\top)$. How does this look in terms of tables?

Recall that to represent an ordinary database instance $I\colon \mathcal{C} \to \mathbf{Set}$, we use a tabular format in which every object $c \in \mathrm{Ob}(\mathcal{C})$ is displayed as a table including one ID column and an additional column for every arrow emanating from c. In the ID column of table c were elements of the set $I(c)$ and in the column assigned to some arrow $f\colon c \to c'$ the cells were elements of the set $I(c')$.

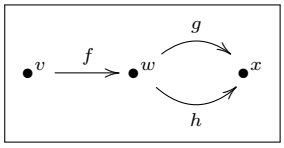
To represent a *Kleisli* database instance $J\colon \mathcal{C} \to \mathbf{Kls}\top$ is similar; we again use a tabular format in which every object $c \in \mathrm{Ob}(\mathcal{C})$ is displayed as a table including one ID column and an additional column for every arrow emanating from c. In the ID column of table c are again elements of the set $J(c)$; however in the column assigned to some arrow $f\colon c \to c'$ are not elements of $J(c')$ but T-values in $J(c')$, i.e. elements of $T(J(c'))$.

Example 5.3.4.1. Let $\top = (T, \eta, \mu)$ be the monad for partial functions, as discussed in Example 5.3.1.1. Given any schema $\mathcal{C}$, we can represent a Kleisli $\top$-instance $I\colon \mathcal{C} \to \mathbf{Kls}(\top)$ in tabular format. To every object $c \in \mathrm{Ob}(\mathcal{C})$ we'll have a set $I(c)$ of rows, and given a column $c \to c'$ every row will produce either a value in $I(c')$ or fail to produce a value; this is the essence of partial functions. We might denote the absence of a value using ☺.

Consider the schema indexing graphs

$$\mathcal{C} := \boxed{\texttt{Arrow} \underset{tgt}{\overset{src}{\rightrightarrows}} \texttt{Vertex}}$$

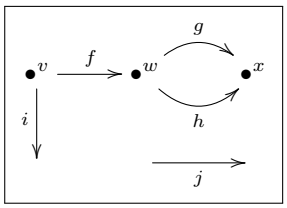
As we discussed in Section 4.2.1.20, an ordinary instance on $\mathcal{C}$ represents a graph.

$I :=$ 

Arrow (I)		
ID	**src**	**tgt**
f	v	w
g	w	x
h	w	x

Vertex (I)
ID
v
w
x

A Kleisli $\top$-instance on $\mathcal{C}$ represents graphs in which edges can fail to have a source vertex, fail to have a target vertex, or both.

$J :=$ 

Arrow (J)		
ID	**src**	**tgt**
f	v	w
g	w	x
h	w	x
i	v	☺
j	☺	☺

Vertex (J)
ID
v
w
x

The context of these tables is that of partial functions, so we do not need a reference for ☺ in the vertex table. Mathematically, the morphism $J(src)\colon J(\texttt{Arrow}) \to J(\texttt{Vertex})$ needs to be a function $J(\texttt{Arrow}) \to J(\texttt{Vertex}) \sqcup \{☺\}$, and it is.

5.3.4.2 Probability distributions

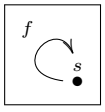
Let $[0,1] \subseteq \mathbb{R}$ denote the set of real numbers between 0 and 1. Let X be a set and $p\colon X \to [0,1]$ a function. We say that p is a *finitary probability distribution on* X if there exists a finite subset $W \subseteq X$ such that

$$\sum_{w \in W} p(w) = 1, \tag{5.12}$$

and such that $p(x) > 0$ if and only if $x \in W$. Note that W is unique if it exists; we call it *the support of* p and denote it $\mathbf{Supp}(p)$. Note also that if X is a finite set then every function p satisfying (5.12) is a finitary probability distribution on X.

For any set X, let $\mathbf{Dist}(X)$ denote the set of finitary probability distributions on X. It is easy to check that given a function $f\colon X \to Y$ one obtains a function $\mathbf{Dist}(f)\colon \mathbf{Dist}(X) \to \mathbf{Dist}(Y)$ by $\mathbf{Dist}(f)(y) = \sum_{f(x)=y} p(x)$. Thus we can consider $\mathbf{Dist}\colon \mathbf{Set} \to \mathbf{Set}$ as a functor, and in fact the functor part of a monad. Its unit $\eta\colon X \to \mathbf{Dist}(X)$ is given by the Kronecker delta function $x \mapsto \delta_x$ where $\delta_x(x) = 1$ and $\delta_x(x') = 0$ for $x' \neq x$. Its multiplication $\mu\colon \mathbf{Dist}(\mathbf{Dist}(X)) \to \mathbf{Dist}(X)$ is given by weighted sum: given a finitary probability distribution $w\colon \mathbf{Dist}(X) \to [0,1]$ and $x \in X$, put $\mu(w)(x) = \sum_{p \in \mathbf{Supp}(w)} w(p)p(x)$.

Example 5.3.4.3 (Markov chains). Let $\mathcal{L}oop$ be the loop schema,

$\mathcal{L}oop :=$ 

as in Example 3.5.2.9. A **Dist**-instance on $\mathcal{L}oop$ is equivalent to a time-homogeneous Markov chain. To be explicit, a functor $\delta\colon \mathcal{L}oop \to \mathbf{KlsDist}$ assigns to the unique object

$s \in \mathrm{Ob}(\mathcal{L}oop)$ a set $S = \delta(s)$, which we call the state space, and to $f\colon s \to s$ a function $\delta(f)\colon S \to \mathbf{Dist}(S)$, which sends each element $x \in S$ to some probability distribution on elements of S. For example, the table δ on the left corresponds to the Markov matrix M on the right below:

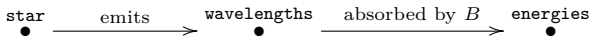
$\delta :=$

s	
ID	**f**
1	.5(1)+.5(2)
2	1(2)
3	.7(1)+.3(3)
4	.4(1)+.3(2)+.3(4)

$$M := \begin{pmatrix} 0.5 & 0.5 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0.7 & 0 & 0.3 & 0 \\ 0.4 & 0.3 & 0 & 0.3 \end{pmatrix} \tag{5.13}$$

As one might hope, for any natural number $n \in \mathbb{N}$ the map $f^n\colon S \to \mathbf{Dist}(S)$ corresponds to the matrix M^n, which sends an element in S to its probable location after n iterations of the transition map.

Application 5.3.4.4. Every star emits a spectrum of light, which can be understood as a distribution on the electromagnetic spectrum. Given an object B on earth, different parts of B will absorb radiation at different rates. Thus B produces a function from the electromagnetic spectrum to distributions of energy absorption. In the context of the probability distributions monad, we can record data on the schema



The composition formula for Kleisli categories is the desired one: to each star we associate the weighted sum of energy absorption rates over the set of wavelengths emitted by the star.

◊◊

5.3.5 Monads and adjunctions

There is a strong connection between monads and adjunctions: every adjunction creates a monad, and every monad "comes from" an adjunction. For example, the List monad (Example 5.3.2.2) comes from the free-forgetful adjunction between sets and monoids

$$\mathbf{Set} \underset{U}{\overset{F}{\rightleftarrows}} \mathbf{Mon}$$

(see Proposition 5.1.1.2). That is, for any set X, the free monoid on X is

$$F(X) = (\mathrm{List}(X), [\,], +\!\!+),$$

and the underlying set of that monoid is $U(F(X)) = \mathrm{List}(X)$. Now it may seem like there was no reason to use monoids at all—the set $\mathrm{List}(X)$ was needed in order to discuss $F(X)$—but it will turn out that the unit η and multiplication μ will come drop out of the adjunction too. First, we discuss the unit and counit of an adjunction.

Definition 5.3.5.1. Let $\mathcal{C}$ and $\mathcal{D}$ be categories, and let $L\colon \mathcal{C} \to \mathcal{D}$ and $R\colon \mathcal{D} \to \mathcal{C}$ be functors with adjunction isomorphism

$$\alpha_{c,d}\colon \mathrm{Hom}_{\mathcal{D}}(L(c), d) \xrightarrow{\cong} \mathrm{Hom}_{\mathcal{C}}(c, R(d))$$

for any objects $c \in \mathrm{Ob}(\mathcal{C})$ and $d \in \mathrm{Ob}(\mathcal{D})$. The *unit* $\eta\colon \mathrm{id}_{\mathcal{C}} \to R \circ L$ (respectively the *counit* $\epsilon\colon L \circ R \to \mathrm{id}_{\mathcal{D}}$) are natural transformations defined as follows.

Given an object $c \in \mathrm{Ob}(\mathcal{C})$, we apply α to $\mathrm{id}_{L(c)}\colon L(c) \to L(c)$ to get

$$\eta_c\colon c \to R \circ L(c);$$

similarly given an object $d \in \mathrm{Ob}(\mathcal{D})$ we apply α^{-1} to $\mathrm{id}_{R(d)}\colon R(d) \to R(d)$ to get

$$\epsilon_d\colon L \circ R(d) \to d.$$

Below we will show how to use the unit and counit of any adjunction to make a monad. We first walk through the process in Example 5.3.5.2.

Example 5.3.5.2. Consider the adjunction $\mathbf{Set} \underset{U}{\overset{F}{\rightleftarrows}} \mathbf{Mon}$ between sets and monoids. Let $T = U \circ F\colon \mathbf{Set} \to \mathbf{Set}$; this will be the functor part of our monad, and we have $T = \mathrm{List}$. Then the unit of the adjunction, $\eta\colon \mathrm{id}_{\mathbf{Set}} \to U \circ F$ is precisely the unit of the monad: for any set $X \in \mathrm{Ob}(\mathbf{Set})$ the component $\eta_X\colon X \to \mathrm{List}(X)$ is the function that takes $x \in X$ to the singleton list $[x] \in \mathrm{List}(X)$. The monad also has a multiplication map $\mu_X\colon T(T(X)) \to T(X)$, which amounts to flattening a list of lists. This function comes about using the counit ϵ, as follows

$$T \circ T = U \circ F \circ U \circ F \xrightarrow{\mathrm{id}_U \diamond \epsilon \diamond \mathrm{id}_F} U \circ F = T.$$

The general procedure for extracting a monad from an adjunction is analogous to that shown in Example 5.3.5.2. Given any adjunction

$$\mathcal{C} \underset{R}{\overset{L}{\rightleftarrows}} \mathcal{D}$$

We define $\top = R \circ L\colon \mathcal{C} \to \mathcal{C}$, we define $\eta\colon \mathrm{id}_{\mathcal{C}} \to \top$ to be the unit of the adjunction (as in Definition 5.3.5.1), and we define $\mu\colon \top \circ \top \to \top$ to be the natural transformation $\mathrm{id}_R \diamond \epsilon \diamond \mathrm{id}_L\colon RLRL \to RL$, obtained by applying the counit $\epsilon\colon LR \to \mathrm{id}_{\mathcal{D}}$.

The above procedure produces monads on arbitrary categories $\mathcal{C}$, whereas our definition of monad (Definition 5.3.2.1) considers only the case $\mathcal{C} = \mathbf{Set}$. However, this definition can be generalized to arbitrary categories $\mathcal{C}$ by simply replacing every occurrence of the string **Set** with the string $\mathcal{C}$. Similarly, our definition of Kleisli categories (Definition 5.3.3.1) considers only the case $\mathcal{C} = \mathbf{Set}$, but again the generalization to arbitrary categories $\mathcal{C}$ is straightforward. In Proposition 5.3.5.3, it may be helpful to again put $\mathcal{C} = \mathbf{Set}$ if one is at all disoriented.

Proposition 5.3.5.3. *Let $\mathcal{C}$ be a category, let $(\top, \eta, \mu)$ be a monad on $\mathcal{C}$, and let $\mathcal{K} := \mathbf{Kls}_{\mathcal{C}}(\top)$ be the Kleisli category. Then there is an adjunction*

$$\mathcal{C} \underset{R}{\overset{L}{\rightleftarrows}} \mathcal{K}$$

such that the monad $(\top, \eta, \mu)$ is obtained (up to isomorphism) by the above procedure.

Sketch of proof. The functor $L\colon \mathcal{C} \to \mathcal{K}$ was discussed in Remark 5.3.3.3. We define it to be identity on objects (recall that $\mathrm{Ob}(\mathcal{K}) = \mathrm{Ob}(\mathcal{C})$). Given objects $c, c' \in \mathrm{Ob}(\mathcal{C})$ the function

$$\mathrm{Hom}_{\mathcal{C}}(c, c') \xrightarrow{L} \mathrm{Hom}_{\mathcal{K}}(c, c') = \mathrm{Hom}_{\mathcal{C}}(c, \top(c'))$$

is given by $f \mapsto \eta_{c'} \circ f$. The fact that this is a functor (i.e. that it preserves composition) follows from a monad axiom.

The functor $R\colon \mathcal{K} \to \mathcal{C}$ acts on objects by sending $c \in \mathrm{Ob}(\mathcal{K}) = \mathrm{Ob}(\mathcal{C})$ to $\top(c) \in \mathrm{Ob}(\mathcal{C})$. For objects $c, c' \in \mathrm{Ob}(\mathcal{K})$ the function

$$\mathrm{Hom}_{\mathcal{C}}(c, \top(c')) = \mathrm{Hom}_{\mathcal{K}}(c, c') \xrightarrow{R} \mathrm{Hom}_{\mathcal{C}}(\top(c), \top(c'))$$

is given by sending the $\mathcal{C}$-morphism $f\colon c \to \top(c')$ to the composite

$$\top(c) \xrightarrow{\top(f)} \top\top(c') \xrightarrow{\mu_{c'}} \top(c').$$

Again, the functoriality follows from monad axioms.

We will not continue on to show that these are adjoint or that they produce the monad $(\top, \eta, \mu)$, but see [Mac, VI.5.1] for the remainder of the proof.

□

Example 5.3.5.4. Let $A \in \mathrm{Ob}(\mathbf{Set})$ be a set, and recall the currying adjunction

$$\mathbf{Set} \underset{-^A}{\overset{A\times -}{\rightleftarrows}} \mathbf{Set}$$

discussed briefly in Example 5.1.1.8. The corresponding monad St_A is typically called the *state monad of type* A in programming language theory. Given a set X, we have

$$St_A(X) = (A \times X)^A.$$

In the Kleisli category $\mathbf{Kls}(St_A)$ a morphism from X to Y is a function of the form $X \to (A \times Y)^A$, but this can be curried to a function $A \times X \to A \times Y$.

This monad is related to holding on to an internal state variable of type A. Every morphism ostensibly from X to Y actually takes as input not only an element of X but also the current state $a \in A$, and it produces as output not only an element of Y but an updated state as well.

Computer scientists in programming language theory have found monads to be very useful ([Mog]). In much the same way, monads on **Set** can be useful in databases, as discussed in Section 5.3.4. Another, totally different way to use monads in databases is by using a mapping between schemas to produce in each one an internal model of the other. That is, for any functor $F\colon \mathcal{C} \to \mathcal{D}$, i.e. mapping of database schemas, the adjunction (Σ_F, Δ_F) produces a monad on $\mathcal{C}$–**Set**, and the adjunction (Δ_F, Π_F) produces a monad on $\mathcal{D}$–**Set**. If one interprets the List monad as producing in **Set** an internal model of the category **Mon** of monoids, one can similarly interpret the above monads on $\mathcal{C}$–**Set** and $\mathcal{D}$–**Set** as producing internal models of each within the other.

5.4 Operads

In this section we briefly introduce operads, which are generalizations of categories. They often are useful for speaking about self-similarity of structure. For example, we will use them to model agents made up of smaller agents, or materials made up of smaller materials. This association with self-similarity is not really inherent in the definition, but it tends to emerge in our thinking about many operads used in practice.

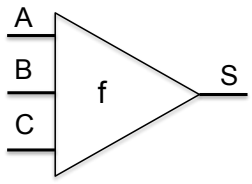
Let me begin with a warning.

Warning 5.4.0.5. My use of the term operad is not entirely standard and conflicts with widespread usage. The more common term for what I am calling an operad is *symmetric colored operad* or a *symmetric multicategory*. An operad classically is a multicategory with one object, and a colored operad is a multicategory. The analogy is that "operad is to multicategory as monoid is to category". The term multicategory stems from the fact that the morphisms in a multicategory have many, rather than one, input. But there is nothing really "multi" about the multicategory itself, only its morphisms. Probably the real reason though is that I find the term multicategory to be clunky and the term operad to be sleek, clocking in at half the syllables. I apologize if my break with standard terminology causes any confusion.

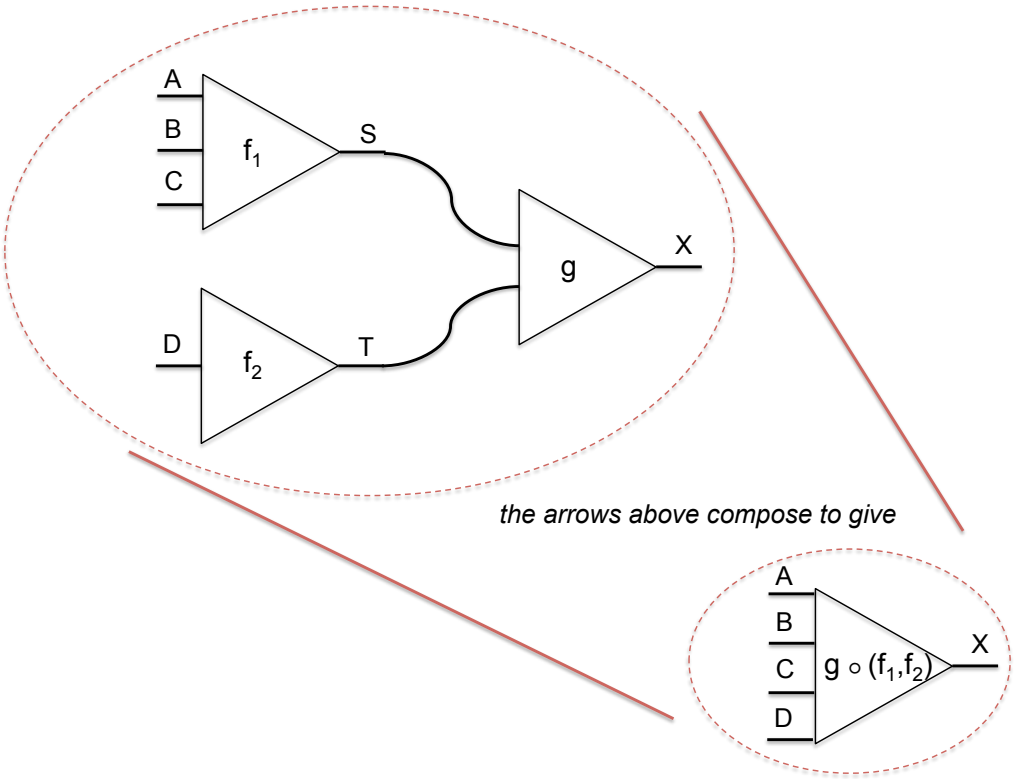
This introduction to operads is quite short. One should see [Le1] for an excellent treatment.

5.4.1 Definition and classical examples

An operad is like a category in that it has objects, morphisms, and a composition formula, and it follows an identity law and an associativity law. The difference is that each morphism has many inputs (and one output).



The description of composition in an operad is a bit heavier than it is in a category, but the idea fairly straightforward. Here is a picture of morphisms being composed.



Note that S and T disappear from the composition, but this is analogous to the way the middle object disappears from the composition of morphisms in a category

$A \xrightarrow{f} S \xrightarrow{g} X$ *the arrows to the left compose to give* $A \xrightarrow{g \circ f} X$

Here is the definition, which we take directly from [Sp4].

Definition 5.4.1.1. An *operad* $\mathcal{O}$ is defined as follows: One announces some constituents (A. objects, B. morphisms, C. identities, D. compositions) and asserts that they conform to some laws (1. identity law, 2. associativity law). Specifically,

A. one announces a collection $\mathrm{Ob}(\mathcal{O})$, each element of which is called an *object* of $\mathcal{O}$.

B. for each object $y \in \mathrm{Ob}(\mathcal{O})$, finite set $n \in \mathrm{Ob}(\mathbf{Fin})$, and n-indexed set of objects $x\colon n \to \mathrm{Ob}(\mathcal{O})$, one announces a set $\mathcal{O}_n(x; y) \in \mathrm{Ob}(\mathbf{Set})$. Its elements are called *morphisms from x to y* in $\mathcal{O}$.

C. for every object $x \in \mathrm{Ob}(\mathcal{O})$, one announces a specified morphism denoted $\mathrm{id}_x \in \mathcal{O}_1(x; x)$ called *the identity morphism on x*.

D. Let $s\colon m \to n$ be a morphism in **Fin**. Let $z \in \mathrm{Ob}(\mathcal{O})$ be an object, let $y\colon n \to \mathrm{Ob}(\mathcal{O})$ be an n-indexed set of objects, and let $x\colon m \to \mathrm{Ob}(\mathcal{O})$ be an m-indexed set of objects. For each element $i \in n$, write $m_i := s^{-1}(i)$ for the pre-image of s under i, and write $x_i = x|_{m_i}\colon m_i \to \mathrm{Ob}(\mathcal{O})$ for the restriction of x to m_i. Then

one announces a function

$$\circ\colon \mathcal{O}_n(y;z) \times \prod_{i\in n} \mathcal{O}_{m_i}(x_i; y(i)) \longrightarrow \mathcal{O}_m(x;z), \tag{5.14}$$

called *the composition formula*.

Given an n-indexed set of objects $x\colon n \to \mathrm{Ob}(\mathcal{O})$ and an object $y \in \mathrm{Ob}(\mathcal{O})$, we sometimes abuse notation and denote the set of morphisms from x to y by $\mathcal{O}(x_1,\dots,x_n;y)$. [15] We may write $\mathrm{Hom}_{\mathcal{O}}(x_1,\dots,x_n;y)$, in place of $\mathcal{O}(x_1,\dots,x_n;y)$, when convenient. We can denote a morphism $\phi \in \mathcal{O}_n(x;y)$ by $\phi\colon x \to y$ or by $\phi\colon (x_1,\dots,x_n) \to y$; we say that each x_i is a *domain object* of ϕ and that y is the *codomain object* of ϕ. We use infix notation for the composition formula, e.g. writing $\psi \circ (\phi_1,\dots,\phi_n)$.

One asserts that the following laws hold:

1. for every $x_1,\dots,x_n,y \in \mathrm{Ob}(\mathcal{O})$ and every morphism $\phi\colon (x_1,\dots,x_n) \to y$, we have
$$\phi \circ (\mathrm{id}_{x_1},\dots,\mathrm{id}_{x_n}) = \phi \qquad \text{and} \qquad \mathrm{id}_y \circ \phi = \phi;$$

2. Let $m \xrightarrow{s} n \xrightarrow{t} p$ be composable morphisms in **Fin**. Let $z \in \mathrm{Ob}(\mathcal{O})$ be an object, let $y\colon p \to \mathrm{Ob}(\mathcal{O})$, $x\colon n \to \mathrm{Ob}(\mathcal{O})$, and $w\colon m \to \mathrm{Ob}(\mathcal{O})$ respectively be a p-indexed, n-indexed, and m-indexed set of objects. For each $i \in p$, write $n_i = t^{-1}(i)$ for the pre-image and $x_i\colon n_i \to \mathrm{Ob}(\mathcal{O})$ for the restriction. Similarly, for each $k \in n$ write $m_k = s^{-1}(k)$ and $w_k\colon m_k \to \mathrm{Ob}(\mathcal{O})$; for each $i \in p$, write $m_{i,-} = (t\circ s)^{-1}(i)$ and $w_{i,-}\colon m_{i,-} \to \mathrm{Ob}(\mathcal{O})$; for each $j \in n_i$, write $m_{i,j} := s^{-1}(j)$ and $w_{i,j}\colon m_{i,j} \to \mathrm{Ob}(\mathcal{O})$. Then the diagram below commutes:

$$\begin{array}{ccc} & \mathcal{O}_p(y;z) \times \prod_{i\in p} \mathcal{O}_{n_i}(x_i;y(i)) \times \prod_{i\in p,\ j\in n_i} \mathcal{O}_{m_{i,j}}(w_{i,j};x_i(j)) & \\ \swarrow & & \searrow \\ \mathcal{O}_n(x;z) \times \prod_{k\in n} \mathcal{O}_{m_k}(w_k;x(k)) & & \mathcal{O}_p(y;z) \times \prod_{i\in p} \mathcal{O}_{m_{i,-}}(w_{i,-};y(i)) \\ & \searrow \quad \swarrow & \\ & \mathcal{O}_m(w;z) & \end{array}$$

Remark 5.4.1.2. In this remark we will discuss the abuse of notation in Definition 5.4.1.1 and how it relates to an action of a symmetric group on each morphism set in our definition of operad. We follow the notation of Definition 5.4.1.1, especially following the use of subscripts in the composition formula.

Suppose that $\mathcal{O}$ is an operad, $z \in \mathrm{Ob}(\mathcal{O})$ is an object, $y\colon n \to \mathrm{Ob}(\mathcal{O})$ is an n-indexed set of objects, and $\phi\colon y \to z$ is a morphism. If we linearly order n, enabling us to write $\phi\colon (y(1),\dots,y(|n|)) \to z$, then changing the linear ordering amounts to finding an isomorphism of finite sets $\sigma\colon m \xrightarrow{\cong} n$, where $|m| = |n|$. Let $x = y \circ \sigma$ and for each $i \in n$, note that $m_i = \sigma^{-1}(\{i\}) = \{\sigma^{-1}(i)\}$, so $x_i = x|_{\sigma^{-1}(i)} = y(i)$. Taking $\mathrm{id}_{x_i} \in \mathcal{O}_{m_i}(x_i;y(i))$ for each $i \in n$, and using the identity law, we find that the composition formula induces a bijection $\mathcal{O}_n(y;z) \xrightarrow{\cong} \mathcal{O}_m(x;z)$, which we might denote by

$$\sigma\colon \mathcal{O}(y(1),y(2),\dots,y(n);z) \cong \mathcal{O}\big(y(\sigma(1)),y(\sigma(2)),\dots,y(\sigma(n));z\big).$$

[15] There are three abuses of notation when writing $\mathcal{O}(x_1,\dots,x_n;y)$, which we will fix one by one. First, it confuses the set $n \in \mathrm{Ob}(\mathbf{Fin})$ with its cardinality $|n| \in \mathbb{N}$. But rather than writing $\mathcal{O}(x_1,\dots,x_{|n|};y)$, it would be more consistent to write $\mathcal{O}(x(1),\dots,x(|n|);y)$, because we have assigned subscripts another meaning in part D. But even this notation unfoundedly suggests that the set n has been endowed with a linear ordering, which it has not. This may be seen as a more serious abuse, but see Remark 5.4.1.2.

In other words, there is an induced group action of $\mathrm{Aut}(n)$ on $\mathcal{O}_n(x;z)$, where $\mathrm{Aut}(n)$ is the group of permutations of an n-element set.

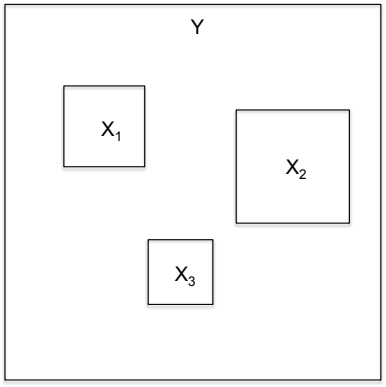
Throughout this book, we will permit ourselves to abuse notation and speak of morphisms $\phi\colon (x_1, x_2, \ldots, x_n) \to y$ for a natural number $n \in \mathbb{N}$, without mentioning the abuse inherent in choosing an order, so long as it is clear that permuting the order of indices would not change anything up to canonical isomorphism.

Example 5.4.1.3. Let **Sets** denote the operad defined as follows. For objects we put $\mathrm{Ob}(\mathbf{Sets}) = \mathrm{Ob}(\mathbf{Set})$. For a natural number $n \in \mathbb{N}$ and sets $X_1, \ldots, X_n, Y$, put

$$\mathrm{Hom}_{\mathbf{Sets}}(X_1, \ldots, X_n; Y) := \mathrm{Hom}_{\mathbf{Set}}(X_1 \times \cdots \times X_n, Y).$$

Given functions $f_1\colon (X_{1,1} \times \cdots \times X_{1,m_1}) \to Y_1$ through $f_n\colon (X_{n,1} \times \cdots \times X_{n,m_n}) \to Y_n$ and a function $Y_1 \times \cdots \times Y_n \to Z$, the universal property provides us a unique function of the form $(X_{1,1} \times \cdots \times X_{n,m_n}) \longrightarrow Z$, giving rise to our composition formula.

Example 5.4.1.4 (Little squares operad). An operad commonly used in mathematics is called the *little n-cubes operad*. We'll focus on $n = 2$ and talk about the little squares operad $\mathcal{O}$. Here the set of objects has only one element, which we denote by a square, $\mathrm{Ob}(\mathcal{O}) = \{\square\}$. For a natural number $n \in \mathbb{N}$, a morphism $f\colon (\square, \square, \ldots, \square) \longrightarrow \square$ is a positioning of n non-overlapping squares inside of a square. Here is a picture of a morphism $(X_1, X_2, X_3) \to Y$, where $X_1 = X_2 = X_3 = Y = \square$.



The composition law says that given a positioning of small squares inside a large square, and given a positioning of tiny squares inside each of those small squares, we get a positioning of tiny squares inside a large square. A picture is shown in Figure 5.15.

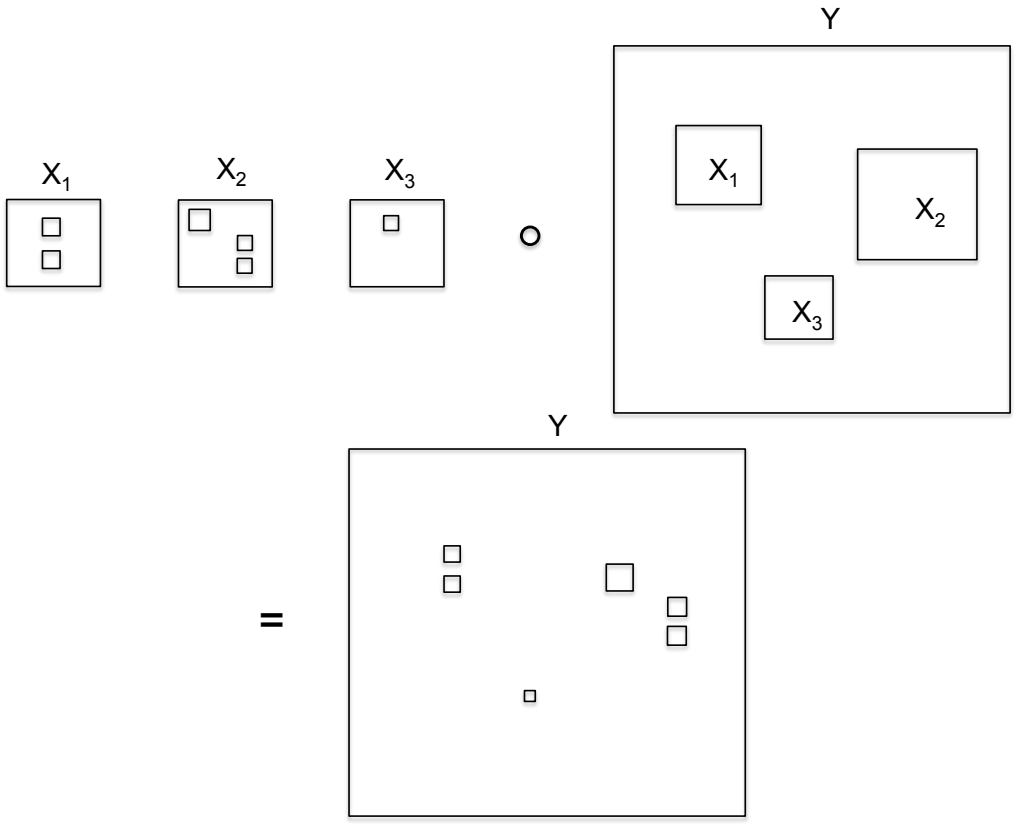


Figure 5.15: Here we show a morphism $(X_1, X_2, X_3) \to Y$ and morphisms $(W_{1,1}, W_{1,2}) \to X_1$, $(W_{2,1}, W_{2,2}, W_{2,3}) \to X_2$, and $(W_{3,1}) \to X_3$, each of which is a positioning of squares inside a square. The composition law scales and positions the squares in the "obvious" way.

Hopefully, what we meant by "self-similarity" in the introduction to this section (see page 247) is becoming clear.

Exercise 5.4.1.5. Consider an operad $\mathcal{O}$ like the little squares operad from Example 5.4.1.4, except with three objects: square, circle, equilateral triangle. A morphism is again a non-overlapping positioning of shapes inside of a shape.

a.) Draw an example of a morphism f from two circles and a square to a triangle.

b.) Find three other morphisms that compose into f, and draw the composite.

◊

5.4.1.6 Operads: functors and algebras

If operads are like categories, then we can define things like functors and call them *operad functors*. Before giving the definition, we give a warning.

Warning 5.4.1.7. What we call operad functors in Definition 5.4.1.8 are usually (if not always) called *operad morphisms*. We thought that the terminology clash between morphisms *of* operads and morphisms *in* an operad was too confusing. It is similar to what would occur in regular category theory (e.g. Chapter 4) if we replaced the term "functor" with the term "category morphism".

Definition 5.4.1.8. Let $\mathcal{O}$ and $\mathcal{O}'$ be operads. An *operad functor from $\mathcal{O}$ to $\mathcal{O}'$*, denoted $F\colon \mathcal{O} \to \mathcal{O}'$ consists of some constituents (A. on-objects part, B. on-morphisms part) conforming to some laws (1. preservation of identities, 2. preservation of composition), as follows:

A. There is a function $\mathrm{Ob}(F)\colon \mathrm{Ob}(\mathcal{O}) \to \mathrm{Ob}(\mathcal{O}')$.

B. For each object $y \in \mathrm{Ob}(\mathcal{O})$, finite set $n \in \mathrm{Ob}(\mathbf{Fin})$, and n-indexed set of objects $x\colon n \to \mathrm{Ob}(\mathcal{O})$, there is a function

$$F_n\colon \mathcal{O}_n(x;y) \to \mathcal{O}'_n(Fx;Fy).$$

As in B. above, we often denote $\mathrm{Ob}(F)$, and also each F_n, simply by F. The laws that govern these constituents are as follows:

1. For each object $x \in \mathrm{Ob}(\mathcal{O})$, the equation $F(\mathrm{id}_x) = \mathrm{id}_{Fx}$ holds.
2. Let $s\colon m \to n$ be a morphism in **Fin**. Let $z \in \mathrm{Ob}(\mathcal{O})$ be an object, let $y\colon n \to \mathrm{Ob}(\mathcal{O})$ be an n-indexed set of objects, and let $x\colon m \to \mathrm{Ob}(\mathcal{O})$ be an m-indexed set of objects. Then, with notation as in Definition 5.4.1.1, the following diagram of sets commutes:

$$\begin{array}{ccc} \mathcal{O}_n(y;z) \times \prod_{i\in n} \mathcal{O}_{m_i}(x_i;y(i)) & \xrightarrow{F} & \mathcal{O}'_n(Fy;Fz) \times \prod_{i\in n} \mathcal{O}'_{m_i}(Fx_i;Fy(i)) \\ \circ\downarrow & & \downarrow\circ \\ \mathcal{O}_m(x;z) & \xrightarrow[F]{} & \mathcal{O}'_m(Fx;Fz) \end{array} \tag{5.16}$$

We denote the category of operads and operad functors by **Oprd**.

Exercise 5.4.1.9. Let $\mathcal{O}$ denote the little squares operad from Example 5.4.1.4 and let $\mathcal{O}'$ denote the operad you constructed in Exercise 5.4.1.5.

a.) Can you come up with an operad functor $\mathcal{O} \to \mathcal{O}'$?

b.) Is it possible to find an operad functor $\mathcal{O}' \to \mathcal{O}$?

◊

Definition 5.4.1.10 (Operad algebra)**.** Let $\mathcal{O}$ be an operad. An *algebra on $\mathcal{O}$* is an operad functor $A\colon \mathcal{O} \to \mathbf{Sets}$.

Remark 5.4.1.11. Every category can be construed as an operad (yes, there is a functor **Cat** $\to$ **Oprd**), by simply not including non-unary morphisms. That is, given a category $\mathcal{C}$, one makes an operad $\mathcal{O}$ with $\mathrm{Ob}(\mathcal{O}) := \mathrm{Ob}(\mathcal{C})$ and with

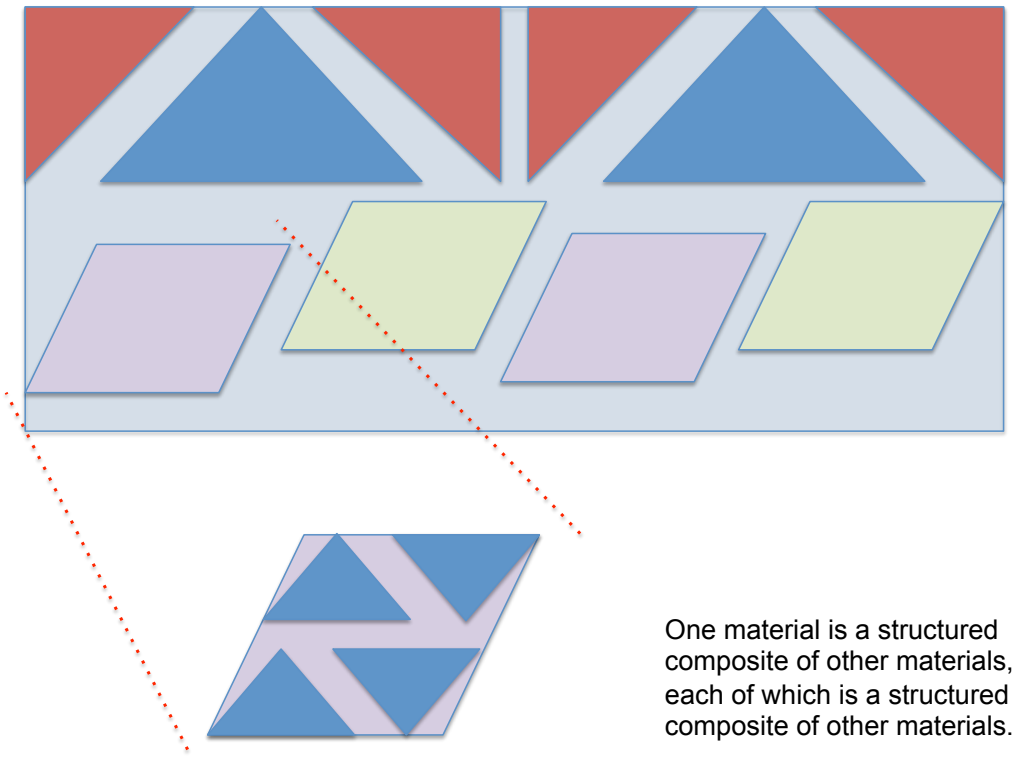
$$\mathrm{Hom}_{\mathcal{O}}(x_1,\ldots,x_n;y) = \begin{cases} \mathrm{Hom}_{\mathcal{C}}(x_1,y) & \text{if } n = 1; \\ \varnothing & \text{if } n \neq 1 \end{cases}$$

Just like a schema is a category presentation, it is possible to discuss operad presentations by generators and relations. Under this analogy, an algebra on an operad corresponds to an instance on a schema.

5.4.2 Applications of operads and their algebras

Hierarchical structures may be well-modeled by operads. Describing such structures using operads and their algebras allows one to make appropriate distinctions between different types of thinking. For example, the allowable formations are encoded in the operad, whereas the elements that will fit into those formations are encoded in the algebra. Morphisms of algebras are high-level understandings of how elements of very different types (such as materials vs. numbers) can occupy the same place in the structure and be compared. We will give examples below.

Application 5.4.2.1. Every material is composed of constituent materials, arranged in certain patterns. (In case the material is "pure", we consider the material to consist of itself as the sole constituent.) Each of these constituent materials each is itself an arrangement of constituent materials. Thus we see a kind of self-similarity which we can model with operads.



(5.17)

For example, a tendon is made of collagen fibers that are assembled in series and then in parallel, in a specific way. Each collagen fibre is made of collagen fibrils that are again assembled in series and then in parallel, with slightly different specifications. We can continue down, perhaps indefinitely, though our resolution fails at some point. A

collagen fibril is made up of tropocollagen collagen molecules, which are twisted ropes of collagen molecules, etc.[16]

Here is how operads might be employed. We want the same operad to model both actual materials, theoretical materials, and functional properties; that is we want more than one algebra on the same operad.

The operad $\mathcal{O}$ should abstractly model the structure, but not the substance being structured. Imagine that each of the shapes (including the background "shape") in Diagram (5.17) is a place-holder, saying something like "*your material here*". Each morphism (that's what (5.17) is a picture of) represents a construction of a material out of parts. In our picture, it appears we are only concerned with the spacial arrangements, but there is far more flexibility than that. Whether we want to allow for additional details beyond spacial arrangements is the kinds of choice we make in a meeting called "what operad should we use?"

◊◊

Application 5.4.2.2. Suppose we have chosen an operad $\mathcal{O}$ to model the structure of materials. Each object of $\mathcal{O}$ might correspond to a certain quality of material, and each morphism corresponds to an arrangement of various qualities to form a new quality. An algebra $A\colon \mathcal{O} \to \mathbf{Sets}$ on $\mathcal{O}$ forces us to choose what substances will fill in for these qualities. For every object $x \in \mathrm{Ob}(\mathcal{O})$, we want a set $A(x)$ which will be the set of materials with that quality. For every arrangement, i.e. morphism, $f\colon (x_1, \ldots, x_n) \to y$, and every choice $a_1 \in A(x_1), \ldots, a_n \in A(x_n)$ of materials, we need to understand what material $a' = A(f)(a_1, \ldots, a_n) \in A(y)$ will emerge when these materials are arranged in accordance with f. We are really pinning ourselves down here.

But there may be more than one interesting algebra on $\mathcal{O}$. Suppose that $B\colon \mathcal{O} \to \mathbf{Sets}$ is an algebra of strengths rather than materials. For each object $x \in \mathrm{Ob}(\mathcal{O})$, which represents some quality, we let $B(x)$ be the set of possible strengths that something of quality x can have. Then for each arrangement, i.e. morphism, $f\colon (x_1, \ldots, x_n) \to y$, and every choice $b_1 \in B(x_1), \ldots, b_n \in B(x_n)$ of strengths, we need to understand what strength $b' = B(f)(b_1, \ldots, b_n) \in B(y)$ will emerge when these strengths are arranged in accordance with f. Certainly an impressive achievement!

Finally, a morphism of algebras $S\colon A \to B$ would consist of a coherent system for assigning to each material $a \in A(X)$ of a given quality x a specific strength $S(a) \in B(X)$, in such a way that morphisms behaved appropriately. In this language we have stated a very precise goal for the field of material mechanics.

◊◊

Exercise 5.4.2.3. Consider again the little squares operad $\mathcal{O}$ from Example 5.4.1.4. Suppose we wanted to use this operad to describe those photographic mosaics.

a.) Come up with an algebra $P\colon \mathcal{O} \to \mathbf{Sets}$ that sends the square to the set of all photos that can be pasted into that square. What does P do on morphisms in $\mathcal{O}$?

b.) Come up with an algebra $C\colon \mathcal{O} \to \mathbf{Sets}$ that sends each square to the set of all colors (visible frequencies of light). In other words, $C(\square)$ is the set of colors, not the set of ways to color the square. What does C do on morphisms in $\mathcal{O}$. Hint: use some kind of averaging scheme for the morphisms.

c.) Guess: if someone were to appropriately define morphisms of $\mathcal{O}$-algebras (something akin to natural transformations between functors $\mathcal{O} \to \mathbf{Sets}$), do you think there

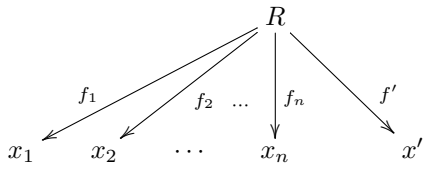
[16] Thanks to Professor Sandra Shefelbine for explaining the hierarchical nature of collagen to me. Any errors are my own.

would some a morphism of algebras $P \to C$?

◊

5.4.2.4 Wiring diagrams

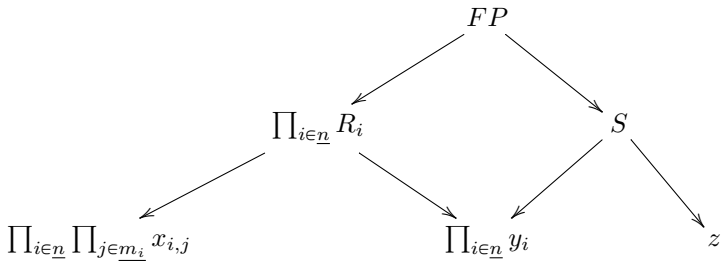
Example 5.4.2.5. Here we describe an *operad of relations*, which we will denote by $\mathcal{R}$. The objects are sets, $\mathrm{Ob}(\mathcal{R}) = \mathrm{Ob}(\mathbf{Set})$. A morphism $f\colon (x_1, x_2, \ldots, x_n) \longrightarrow x'$ in $\mathcal{R}$ is a diagram in **Set** of the form



(5.18)

such that the induced function $R \longrightarrow (x_1 \times x_2 \times \cdots \times x_n \times x')$ is an injection.

We use a composition formula similar to that in Definition 2.5.2.3. Namely, we form a fiber product

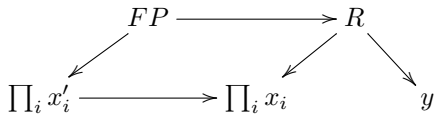


One can show that the induces function $FP \longrightarrow \left(\prod_{i\in\underline{n}} \prod_{j\in\underline{m_i}} x_i\right) \times y$ is an injection, so we have a valid composition formula. Finally, the associativity and identity laws hold. [17]

Application 5.4.2.6. Suppose we are trying to model life in the following way. We define an entity as a set of phenomena, but in order to use colloquial language we say the entity *is able to experience* that set of phenomena. We also want to be able to put entities together to form a super-entity, so we have a notion of morphism $f\colon (e_1, \ldots, e_n) \longrightarrow e'$ defined as a relation as in (5.18). The idea is that the morphism f is a way of translating between the phenomena that may be experienced by the sub-entities and the phenomena that may be experienced by the super-entity.

The operad $\mathcal{R}$ from Example 5.4.2.5 becomes useful as a language for discussing issues in this domain. ◊◊

Example 5.4.2.7. Let $\mathcal{R}$ be the operad of relations from Example 5.4.2.5. Consider the algebra $S\colon \mathcal{R} \to \mathbf{Sets}$ given by $S(x) = \mathbb{P}(x)$. Given a morphism $\prod_i x_i \leftarrow R \to y$ and subsets $x'_i \subseteq x_i$, we have a subset $\prod_i x'_i \subseteq \prod_i x_i$. We take the fiber product



[17] Technically we need to use isomorphism classes of cone points, but we don't worry about this here.

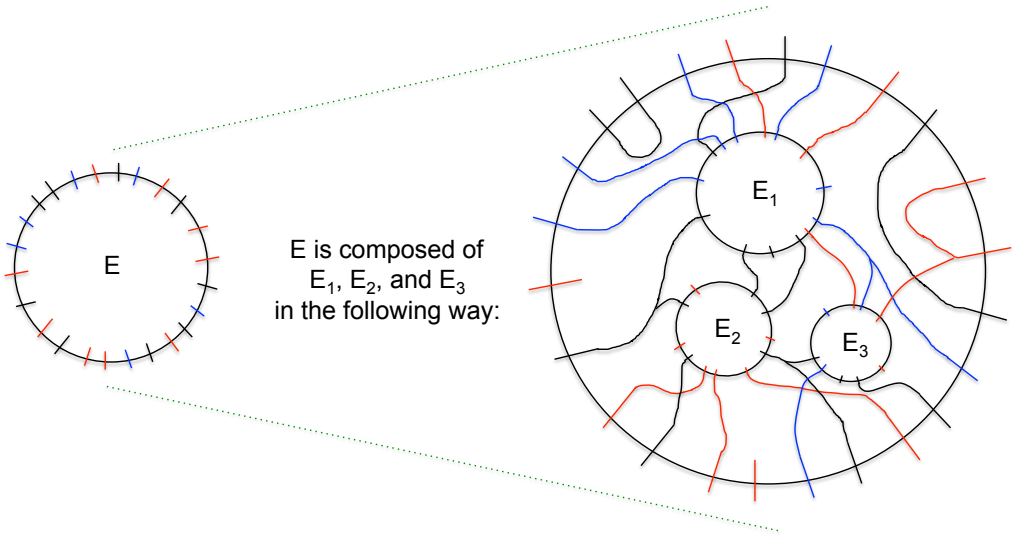
and the image of $FP \to y$ is a subset of y.

Application 5.4.2.8. Following Application 5.4.2.6 we can use Example 5.4.2.7 as a model of survival. Each entity survives only for a subset of the phenomena that it can experience. Under this interpretation, the algebra from Example 5.4.2.7 defines survival as the survival of all parts. That is, suppose that we understand how a super-entity is composed of sub-entities in the sense that we have a translation between the set of phenomena that may be experienced across the sub-entities and the set of phenomena that may be experienced by the super-entity. Then the super-entity will survive exactly those phenomena which translate to phenomena for which each sub-entity desires.

Perhaps a better term than survival would be "allowance". A bureaucracy consists of a set of smaller bureaucracies, each of which allows certain phenomena to pass; the whole bureaucracy allows something to pass if and only if, when translated to the perspective of each sub-bureaucracy, it is allowed to pass there.

◊◊

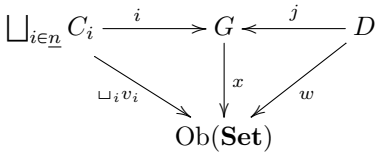
Example 5.4.2.9. In this example we discuss wiring diagrams that look like this:



The operad in question will be denoted $\mathcal{W}$; it is discussed in greater detail in [Sp4]. The objects of $\mathcal{W}$ are pairs (C, s) where C is a finite set and $v\colon C \to \mathrm{Ob}(\mathbf{Set})$ is a function. Think of such an object as a circle with C-many cables sticking out of it; each cable c is assigned a set $v(c)$ corresponding to the set of values that can be carried on that cable. For example $E_2 = (C, v)$ where $|C| = 11$ and we consider v to be specified by declaring that black wires carry $\mathbb{Z}$ and red wires carry {sweet, sour, salty, bitter, umami}.

The morphisms in $\mathcal{W}$ will be pictures as above, formalized as follows. Given objects $(C_1, v_1), \ldots, (C_n, v_n), (D, w)$, a morphism $F\colon ((C_1, v_1), \ldots, (C_n, v_n)) \longrightarrow (D, w)$ is

a commutative diagram of sets [18]



such that i and j are jointly surjective.

Composition of morphisms is easily understood in pictures: given wiring diagrams inside of wiring diagrams, we can throw away the intermediary circles. In terms of sets, we perform a pushout.

There is an operad functor $\mathcal{W} \to \mathcal{S}$ given by sending (C, v) to $\prod_{c\in C} v(c)$. The idea is that to an entity defined as having a bunch of cables carrying variables, a phenomenon is the same thing as a choice of value on each cable. A wiring diagram translates between values experienced locally and values experienced globally.

Application 5.4.2.10. In cognitive neuroscience or in industrial economics, it may be that we want to understand the behavior of an entity such as a mind, a society, or a business in terms of its structure. Knowing the connection pattern (connectome, supply chain) of sub-entities should help us understand how big changes are generated from small ones.

Under the functor $\mathcal{W} \to \mathcal{S}$ the algebra $\mathcal{S} \to$ **Sets** from Application 5.4.2.8 becomes an algebra $\mathcal{W} \to$ **Sets**. To each entity we now associate some subset of the value-assignments it can carry. ◊◊

Application 5.4.2.11. In [RS], Radul and Sussman discuss propagator networks. These can presumably be understood in terms of wiring diagrams and their algebra of relations. ◊◊

[18] If one is concerned with cardinality issues, fix a cardinality κ and replace Ob(**Set**) everywhere with Ob($\mathbf{Set}_{<\kappa}$).

Index

Bibliography

[Ati] Atiyah, M. (1989) "Topological quantum field theories". *Publications Mathématiques de l'IHÉS* 68 (68), pp. 175–186.

[Axl] Axler, S. (1997) *Linear algebra done right.* Springer.

[Awo] S. Awodey. (2010) *Category theory.* Second edition. Oxford Logic Guides, 52. Oxford University Press, Oxford.

[Bar] Bralow, H. (1961) "Possible principles underlying the transformation of sensory messages". *Sensory communication*, pp. 217 – 234.

[BD] Baez, J.C.; Dolan, J. (1995) "Higher-dimensional algebra and topological quantum field theory". *Journal of mathematical physics* vol 36, 6073.

[BFL] Baez, J.C.; Fritz, T.; Leinster, T. (2011) "A characterization of entropy in terms of information loss." *Entropy* 13, no. 11.

[BS] Baez, J.C.; Stay, M. (2011) "Physics, topology, logic and computation: a Rosetta Stone." *New structures for physics*, 95Ð172. Lecture Notes in Phys., 813, Springer, Heidelberg.

[BP1] Brown, R.; Porter, T. (2006) "Category Theory: an abstract setting for analogy and comparison, In: *What is Category Theory?* Advanced Studies in Mathematics and Logic, Polimetrica Publisher, Italy, pp. 257-274.

[BP2] Brown, R.; Porter, T. (2003) "Category theory and higher dimensional algebra: potential descriptive tools in neuroscience", *Proceedings of the International Conference on Theoretical Neurobiology, Delhi*, edited by Nandini Singh, National Brain Research Centre, Conference Proceedings 1 80-92.

[BW] M. Barr, C. Wells. (1990) *Category theory for computing science.* Prentice Hall International Series in Computer Science. Prentice Hall International, New York.

[Big] Biggs, N.M. (2004) *Discrete mathematics.* Oxford University Press, NY.

[Dia] Diaconescu, R. (2008) *Institution-independent model theory* Springer.

[DI] Döring, A.; Isham, C. J. "A topos foundation for theories of physics. I. Formal languages for physics." *J. Math. Phys.* 49 (2008), no. 5, 053515.

[EV] Ehresmann, A.C.; Vanbremeersch, J.P. (2007) *Memory evolutive systems; hierarchy, emergence, cognition.* Elsevier.

[Eve] Everett III, H. (1973). "The theory of the universal wave function." In *The many-worlds interpretation of quantum mechanics* (Vol. 1, p. 3).

[Gog] Goguen, J. (1992) "Sheaf semantics for concurrent interacting objects" *Mathematical structures in Computer Science* Vol 2, pp. 159 – 191.

[Gro] Grothendieck, A. (1971). *Séminaire de Géométrie Algébrique du Bois Marie - 1960-61 - Revêtements étales et groupe fondamental - (SGA 1)* (Lecture notes in mathematics 224) (in French). Berlin; New York: Springer-Verlag.

[Kro] Krömer, R. (2007). *Tool and Object: A History and Philosophy of Category Theory*, Birkhauser.

[Lam] Lambek, J. (1980) "From λ-calculus to Cartesian closed categories". In *Formalism*, Academic Press, London, pp. 375 – 402.

[Law] Lawvere, F.W. (2005) "An elementary theory of the category of sets (long version) with commentary." (Reprinted and expanded from Proc. Nat. Acad. Sci. U.S.A. **52** (1964)) *Repr. Theory Appl. Categ.* 11, pp. 1 – 35.

[Kho] Khovanov, M. (2000) "A categorificiation of the Jones polynomial" *Duke Math J.*.

[Le1] Leinster, T. (2004) *Higher Operads, Higher Categories.* London Mathematical Society Lecture Note Series 298, Cambridge University Press.

[Le2] Leinster, T. (2012) "Rethinking set theory". ePrint available http://arxiv.org/abs/1212.6543.

[Lin] Linsker, R. (1988) "Self-organization in a perceptual network". *Computer* 21, no. 3, pp. 105 – 117.

[LM] Landry, E.; Marquis, J-P., 2005, "Categories in Contexts: historical, foundational, and philosophical." *Philosophia Mathematica*, (3), vol. 13, no. 1, 1 – 43.

[LS] F.W. Lawvere, S.H. Schanuel. (2009) *Conceptual mathematics. A first introduction to categories.* Second edition. Cambridge University Press, Cambridge.

[MacK] MacKay, D.J. (2003). *Information theory, inference and learning algorithms.* Cambridge university press.

[Mac] Mac Lane, S. (1998) *Categories for the working mathematician.* Second edition. Graduate Texts in Mathematics, 5. Springer-Verlag, New York.

[Mar1] Marquis, J-P. (2009) *From a Geometrical Point of View: a study in the history and philosophy of category theory*, Springer.

[Mar2] Marquis, J-P, "Category Theory", *The Stanford Encyclopedia of Philosophy* (Spring 2011 Edition), Edward N. Zalta (ed.), http://plato.stanford.edu/archives/spr2011/entries/category-theory

[Min] Minsky, M. *The Society of Mind.* Simon and Schuster, NY 1985.

[Mog] Moggi, E. (1989) "A category-theoretic account of program modules." *Category theory and computer science (Manchester, 1989),* 101Ð117, Lecture Notes in Comput. Sci., 389, Springer, Berlin.

[nLa] nLab authors. http://ncatlab.org/nlab/show/HomePage

[Pen] Penrose, R. (2006) *The road to reality.* Random house.

[RS] Radul, A.; Sussman, G.J. (2009). "The art of the propagator". *MIT Computer science and artificial intelligence laboratory technical report.*

[Sp1] Spivak, D.I. (2012) "Functorial data migration". *Information and communication*

[Sp2] Spivak, D.I. (2012) "Queries and constraints via lifting problems". Submitted to *Mathematical structures in computer science.* ePrint available: http://arxiv.org/abs/1202.2591

[Sp3] Spivak, D.I. (2012) "Kleisli database instances". ePrint available: http://arxiv.org/abs/1209.1011

[Sp4] Spivak, D.I. (2013) "The operad of wiring diagrams: Formalizing a graphical language for databases, recursion, and plug-and-play circuits". Available online: http://arxiv.org/abs/1305.0297

[SGWB] Spivak D.I., Giesa T., Wood E., Buehler M.J. (2011) "Category Theoretic Analysis of Hierarchical Protein Materials and Social Networks." PLoS ONE 6(9): e23911. doi:10.1371/journal.pone.0023911

[SK] Spivak, D.I., Kent, R.E. (2012) "Ologs: A Categorical Framework for Knowledge Representation." *PLoS ONE* 7(1): e24274. doi:10.1371/journal.pone.0024274.

[WeS] Weinberger, S. (2011) "What is... Persistent Homology?" AMS.

[WeA] Weinstein, A. (1996) "Groupoids: unifying internal and external symmetry. *Notices of the AMS* Vol 43, no. 7, pp. 744 – 752.

[Wik] Wikipedia (multiple authors). Various articles, all linked with a hyperreference are scattered throughout this text. All accessed December 6, 2012 – September 17, 2013.

18.S996 Category Theory for Scientist

Spring 2013