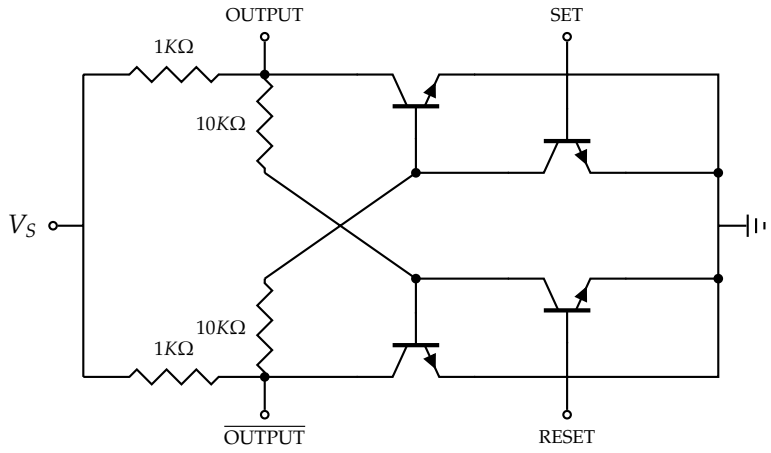
Chapter 6

Electric circuits: Hypergraph categories and operads

6.1 The ubiquity of network languages

Electric circuits, chemical reaction networks, finite state automata, Markov processes: these are all models of physical or computational systems that are commonly described using network diagrams. Here, for example, we draw a diagram that models a flip-flop, an electric circuit—important in computer memory—that can store a bit of information:



Network diagrams have time-tested utility. In this chapter, we are interested in understanding the common mathematical structure that they share, for the purposes of translating between and unifying them; for example certain types of Markov processes can be simulated and hence solved using circuits of resisters. When we understand the underlying structures that are shared by network diagram languages, we can make comparisons between the corresponding mathematical models easily.

At first glance network diagrams appear quite different from the wiring diagrams we have seen so far. For example, the wires are undirected in the case above, whereas in a

category—including monoidal categories seen in resource theories or co-design—every morphism has a domain and codomain, giving it a sense of direction. Nonetheless, we shall see how to use categorical constructions such as universal properties to create categorical models that precisely capture the above type of "network" compositionality, i.e. allowing us to effectively drop directedness when convenient.

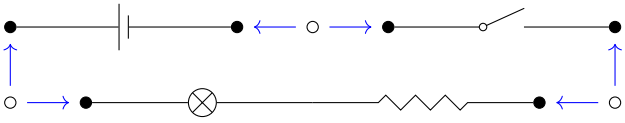
In particular we'll return to the idea of a colimit, which we sketched for you at the end of Chapter 3, and show how to use colimits in the category of sets to formalize ideas of connection. Here's the key idea.

Connections via colimits. Let's say we want to install some lights: we want to create a circuit so that when we flick a switch, a light turns on or off. To start, we have a bunch of circuit components: a power source, a switch, and a lamp connected to a resistor:

We want to connect them together, but there are many ways to do so. How should we describe the particular way that will form a light switch?

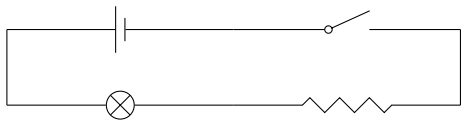
First, we claim that circuits should really be thought of as open circuits: each carries the additional structure of an 'interface' exposing it to the rest of the electrical world. Here by *interface* we mean a certain set of locations, or *ports*, at which we are able to connect them with other components.[1] As is so common in category theory, we begin by making this more-or-less obvious fact explicit. Let's depict the available ports using a bold •. If we say that in the each of the three drawings above, the ports are simply the dangling end points of the wires, they would be redrawn as follows:

Next, we have to describe which ports should be connected. We'll do this by drawing empty circles ∘ connected by arrows to two ports •. Each will be a witness-to-connection, saying 'connect these two!'



[1] If your circuit has no such ports, it still falls within our purview, by taking its interface to be the empty set.

Looking at this picture, it is clear what we need to do: just identify—i.e. *merge* or *make equal*—the ports as indicated, to get the following circuit:



But mathematics doesn't have a visual cortex with which to generate the intuitions we can count on with a human reader such as yourself.[2] Thus we need to specify formally what 'identifying ports as indicated' means mathematically. As it turns out, we can do this using finite colimits in a given category $\mathcal{C}$.

Colimits are diagrams with certain universal properties, which is kind of an epiphenomenon of the category $\mathcal{C}$. Our goal is to obtain $\mathcal{C}$'s colimits more directly, as a kind of operation in some context, so that we can think of them as telling us how to connect circuit parts together. To that end, we produce a certain monoidal category—namely that of *cospans in* $\mathcal{C}$, denoted $\mathbf{Cospan}_{\mathcal{C}}$—that can conveniently package $\mathcal{C}$'s colimits in terms of its own basic operations: composition and monoidal product.

In summary, the first part of this chapter is devoted to the slogan 'colimits model interconnection'. In addition to universal constructions such as colimits, however, another way to describe interconnection is to use wiring diagrams. We go full circle when we find that these wiring diagrams are strongly connected to cospans, and hence colimits.

Composition operations and wiring diagrams. In this book we have seen the utility of defining syntactic or algebraic structures that describe the sort of composition operations that make sense and can be performed in a given application area. Examples include monoidal preorders with discarding, props, and compact closed categories. Each of these has an associated sort of wiring diagram style, so that any wiring diagram of that style represents a composition operation that makes sense in the given area: the first makes sense in manufacturing, the second in signal flow, and the third in collaborative design. So our second goal is to answer the question, "how do we describe the compositional structure of network-style wiring diagrams?"

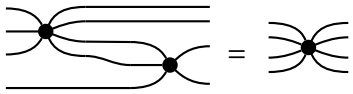
Network-type interconnection can be described using something called a hypergraph category. Roughly speaking, these are categories whose wiring diagrams are those of symmetric monoidal categories together with, for each pair of natural numbers (m, n), an icon $s_{m,n} : m \to n$. These icons, known as *spiders*,[3] are drawn as follows:

Two spiders can share a leg, and when they do, we can fuse them into one spider. The intuition is that spiders are connection points for a number of wires, and when two

[2] Unless the future has arrived since the writing of this book.

[3] Our spiders have any number of legs.

connection points are connected, they fuse to form an even more 'connect-y' connection point. Here is an example:



A hypergraph category may have many species of spiders with the rule that spiders of different species cannot share a leg—and hence not fuse—but two spiders of the same species can share legs and fuse. We add spider diagrams to the iconography of hypergraph categories.

As we shall see, the ideas of describing network interconnection using colimits and hypergraph categories come together in the notion of a theory. We first introduced the idea of a theory in Section 5.4.2, but here we explore it more thoroughly, starting with the idea that, approximately speaking, cospans in the category **FinSet** form the theory of hypergraph categories.

We can assemble all cospans in **FinSet** into something called an 'operad'. Throughout this book we have talked about using free structures and presentations to create instances of algebraic structures such as preorders, categories, and props, tailored to the needs of a particular situation. Operads can be used to tailor the algebraic structures *themselves* to the needs of a particular situation. We will discuss how this works, in particular how operads encode various sorts of wiring diagram languages and corresponding algebraic structures, at the end of the chapter.

6.2 Colimits and connection

Universal constructions are central to category theory. They allow us to define objects, at least up to isomorphism, by describing their relationship with other objects. So far we have seen this theme in a number of different forms: meets and joins (Section 1.3), Galois connections and adjunctions (Sections 1.4 and 3.4), limits (Section 3.5), and free and presented structures (Section 5.2.3-5.2.5). Here we turn our attention to colimits.

In this section, our main task is to have a concrete understanding of colimits in the category **FinSet** of finite sets and functions. The idea will be to take a bunch of sets—say two or fifteen or zero—use functions between them to designate that elements in one set 'should be considered the same' as elements in another set, and then merge the sets together accordingly.

6.2.1 Initial objects

Just as the simplest limit is a terminal object (see Section 3.5.1), the simplest colimit is an initial object. This is the case where you start with no objects and you merge them together.

Definition 6.1. Let $\mathcal{C}$ be a category. An *initial object* in $\mathcal{C}$ is an object $\varnothing \in \mathcal{C}$ such that for each object T in $\mathcal{C}$ there exists a unique morphism $!_T : \varnothing \to T$.

The symbol $\varnothing$ is just a default name, a notation, intended to evoke the right idea; see Example 6.4 for the reason why we use the notation $\varnothing$, and Exercise 6.7 for a case when the default name $\varnothing$ would probably not be used.

Again, the hallmark of universality is the existence of a unique map to any other comparable object.

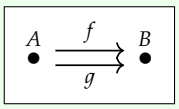
Example 6.2. An initial object of a preorder is a bottom element—that is, an element that is less than every other element. For example 0 is the initial object in $(\mathbb{N}, \leq)$, whereas $(\mathbb{R}, \leq)$ has no initial object.

Exercise 6.3. Consider the set $A = \{a, b\}$. Find a preorder relation $\leq$ on A such that

1. $(A, \leq)$ has no initial object.
2. $(A, \leq)$ has exactly one initial object.
3. $(A, \leq)$ has two initial objects. ◊

Example 6.4. The initial object in **FinSet** is the empty set. Given any finite set T, there is a unique function $\varnothing \to T$, since $\varnothing$ has no elements.

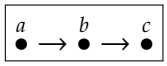
Example 6.5. As seen in Exercise 6.3, a category $\mathcal{C}$ need not have an initial object. As a different sort of example, consider the category shown here:

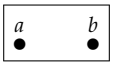
$\mathcal{C} :=$ 

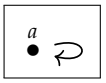
If there were to be an initial object $\varnothing$, it would either be A or B. Either way, we need to show that for each object $T \in \mathrm{Ob}(\mathcal{C})$ (i.e. for both $T = A$ and $T = B$) there is a unique morphism $\varnothing \to T$. Trying the case $\varnothing =^? A$ this condition fails when $T = B$: there are two morphisms $A \to B$, not one. And trying the case $\varnothing =^? B$ this condition fails when $T = A$: there are zero morphisms $B \to A$, not one.

Exercise 6.6. For each of the graphs below, consider the free category on that graph, and say whether it has an initial object.

1.

2. 

3. 

4. 

◊

Exercise 6.7. Recall the notion of rig from Chapter 5. A *rig homomorphism* from $(R, 0_R, +_R, 1_R, *_R)$ to $(S, 0_S, +_S, 1_S, *_S)$ is a function $f : R \to S$ such that $f(0_R) = 0_S$, $f(r_1 +_R r_2) = f(r_1) +_S f(r_2)$, etc.

1. We said "etc." Guess the remaining conditions for f to be a rig homomorphism.

2. Let **Rig** denote the category whose objects are rigs and whose morphisms are rig homomorphisms. We claim **Rig** has an initial object. What is it? ◊

Exercise 6.8. Explain the statement "the hallmark of universality is the existence of a unique map to any other comparable object," in the context of Definition 6.1. In particular, what is being universal in Definition 6.1, and which is the "comparable object"? ◊

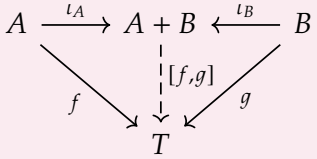
Remark 6.9. As mentioned in Remark 3.85, we often speak of 'the' object that satisfies a universal property, such as 'the initial object', even though many different objects could satisfy the initial object condition. Again, the reason is that initial objects are unique up to unique isomorphism: any two initial objects will have a canonical isomorphism between them, which one finds using various applications of the universal property.

Exercise 6.10. Let $\mathcal{C}$ be a category, and suppose that c_1 and c_2 are initial objects. Find an isomorphism between them, using the universal property from Definition 6.1. ◊

6.2.2 Coproducts

Coproducts generalize both joins in a preorder and disjoint unions of sets.

Definition 6.11. Let A and B be objects in a category $\mathcal{C}$. A *coproduct* of A and B is an object, which we denote $A + B$, together with a pair of morphisms $(\iota_A \colon A \to A + B, \iota_B \colon B \to A + B)$ such that for all objects T and pairs of morphisms $(f \colon A \to T, g \colon B \to T)$, there exists a unique morphism $[f, g] \colon A + B \to T$ such that the following diagram commutes:



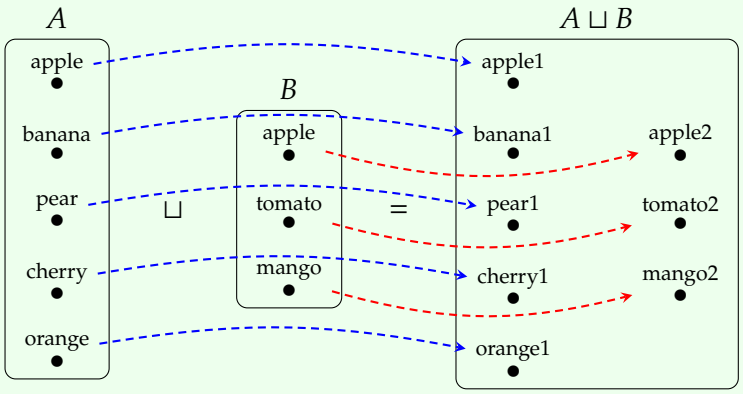
(6.12)

We call $[f, g]$ the *copairing* of f and g.

Exercise 6.13. Explain why, in a preorder, coproducts are the same as joins. ◊

Example 6.14. Coproducts in the categories **FinSet** and **Set** are disjoint unions. More precisely, suppose A and B are sets. Then the coproduct of A and B is given by the disjoint union $A \sqcup B$ together with the inclusion functions $\iota_A \colon A \longrightarrow A \sqcup B$ and

$\iota_B : B \to A \sqcup B$.



(6.15)

Suppose we have functions $f : A \to T$ and $g : B \to T$ for some other set T, unpictured. The universal property of coproducts says there is a unique function $[f, g] : A \sqcup B \to T$ such that $\iota_A \fatsemi [f, g] = f$ and $\iota_B \fatsemi [f, g] = g$. What is it? Any element $x \in A \sqcup B$ is either 'from A' or 'from B', i.e. either there is some $a \in A$ with $x = \iota_A(a)$ or there is some $b \in B$ with $x = \iota_B(b)$. By Eq. (6.12), we must have:

$$[f, g](x) = \begin{cases} f(x) & \text{if } x = \iota_A(a) \text{ for some } a \in A; \\ g(x) & \text{if } x = \iota_B(b) \text{ for some } b \in B. \end{cases}$$

Exercise 6.16. Suppose $T = \{a, b, c, \dots, z\}$ is the set of letters in the alphabet, and let A and B be the sets from Eq. (6.15). Consider the function $f : A \to T$ sending each element of A to the first letter of its label, e.g. $f(\text{apple}) = a$. Let $g : B \to T$ be the function sending each element of B to the last letter of its label, e.g. $g(\text{apple}) = e$. Write down the function $[f, g](x)$ for all eight elements of $A \sqcup B$. ◊

Exercise 6.17. Let $f : A \to C$, $g : B \to C$, and $h : C \to D$ be morphisms in a category $\mathcal{C}$ with coproducts. Show that

1. $\iota_A \fatsemi [f, g] = f$.
2. $\iota_B \fatsemi [f, g] = g$.
3. $[f, g] \fatsemi h = [f \fatsemi h, g \fatsemi h]$.
4. $[\iota_A, \iota_B] = \mathrm{id}_{A+B}$. ◊

Exercise 6.18. Suppose a category $\mathcal{C}$ has coproducts, denoted $+$, and an initial object, denoted $\varnothing$. Then $(\mathcal{C}, +, \varnothing)$ is a symmetric monoidal category (recall Definition 4.45). In this exercise we develop the data relevant to this fact:

1. Show that $+$ extends to a functor $\mathcal{C} \times \mathcal{C} \to \mathcal{C}$. In particular, how does it act on morphisms in $\mathcal{C} \times \mathcal{C}$?
2. Using the universal properties of the initial object and coproduct, show that there are isomorphisms $A + \varnothing \to A$ and $\varnothing + A \to A$.
3. Using the universal property of the coproduct, write down morphisms
   a) $(A + B) + C \to A + (B + C)$.
   b) $A + B \to B + A$.

If you like, check that these are isomorphisms.
It can then be checked that this data obeys the axioms of a symmetric monoidal category, but we'll end the exercise here. ◊

6.2.3 Pushouts

Pushouts are a way of combining sets. Like a union of subsets, a pushout can combine two sets in a non-disjoint way: elements of one set may be identified with elements of the other. The pushout construction, however, is much more general: it allows (and requires) the user to specify exactly which elements will be identified. We'll see a demonstration of this additional generality in Example 6.29.

Definition 6.19. Let $\mathcal{C}$ be a category and let $f\colon A \to X$ and $g\colon A \to Y$ be morphisms in $\mathcal{C}$ that have a common domain. The *pushout* $X +_A Y$ is the colimit of the diagram

$$\begin{array}{ccc} A & \xrightarrow{f} & X \\ {\scriptstyle g}\downarrow & & \\ Y & & \end{array}$$

In more detail, a pushout consists of (i) an object $X +_A Y$ and (ii) morphisms $\iota_X\colon X \to X +_A Y$ and $\iota_Y\colon Y \to X +_A Y$ satisfying (a) and (b) below.

(a) The diagram

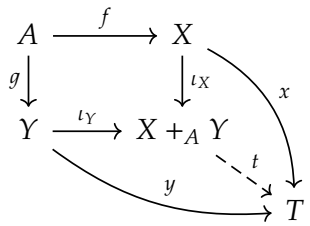
$$\begin{array}{ccc} A & \xrightarrow{f} & X \\ {\scriptstyle g}\downarrow & \ulcorner & \downarrow{\scriptstyle \iota_X} \\ Y & \xrightarrow[\iota_Y]{} & X +_A Y \end{array} \tag{6.20}$$

commutes. (We will explain the '$\ulcorner$' symbol below.)

(b) For all objects T and morphisms $x\colon X \to T$, $y\colon Y \to T$, if the diagram

$$\begin{array}{ccc} A & \xrightarrow{f} & X \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle x} \\ Y & \xrightarrow[y]{} & T \end{array}$$

commutes, then there exists a unique morphism $t\colon X +_A Y \to T$ such that



(6.21)

commutes.

If $X +_A Y$ is a pushout, we denote that fact by drawing the commutative square Eq. (6.20), together with the $\ulcorner$ symbol as shown; we call it a *pushout square*.

We further call ι_X the *pushout of g along f*, and similarly ι_Y the *pushout of f along g*.

Example 6.22. In a preorder, pushouts and coproducts have a lot in common. The pushout of a diagram $B \leftarrow A \rightarrow C$ is equal to the coproduct $B \sqcup C$: namely, both are equal to the join $B \vee C$.

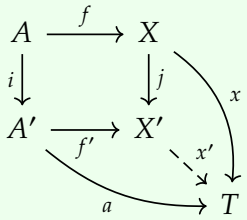
Example 6.23. Let $f : A \to X$ be a morphism in a category $\mathcal{C}$. For any isomorphisms $i : A \to A'$ and $j : X \to X'$, we can take X' to be the pushout $X +_A A'$, i.e. the following is a pushout square:

$$\begin{array}{ccc} A & \xrightarrow{f} & X \\ {\scriptstyle i}\downarrow & \ulcorner & \downarrow{\scriptstyle j} \\ A' & \xrightarrow[f']{} & X' \end{array}$$

where $f' \coloneqq i^{-1} \fatsemi f \fatsemi j$. To see this, observe that if there is any object T such that the following square commutes:

$$\begin{array}{ccc} A & \xrightarrow{f} & X \\ {\scriptstyle i}\downarrow & & \downarrow{\scriptstyle x} \\ A' & \xrightarrow[a]{} & T \end{array}$$

then $f \fatsemi x = i \fatsemi a$, and so we are forced to take $x' : X \to T$ to be $x' \coloneqq j^{-1} \fatsemi x$. This makes the following diagram commute:



because $f' \fatsemi x' = i^{-1} \fatsemi f \fatsemi j \fatsemi j^{-1} \fatsemi x = i^{-1} \fatsemi i \fatsemi a = a$.

Exercise 6.24. For any set S, we have the discrete category $\mathbf{Disc}_S$, with S as objects and only identity morphisms.

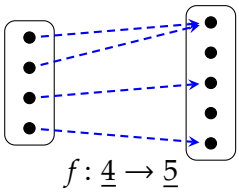
1. Show that all pushouts exist in $\mathbf{Disc}_S$, for any set S.
2. For what sets S does $\mathbf{Disc}_S$ have an initial object? ◊

Example 6.25. In the category **FinSet**, pushouts always exist. The pushout of functions $f : A \to X$ and $g : A \to Y$ is the set of equivalence classes of $X \sqcup Y$ under the equivalence relation generated by—that is, the reflexive, transitive, symmetric closure of—the

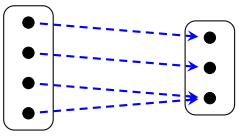
relation $\{f(a) \sim g(a) \mid a \in A\}$.

We can think of this in terms of interconnection too. Each element $a \in A$ provides a connection between $f(a)$ in X and $g(a)$ in Y. The pushout is the set of connected components of $X \sqcup Y$.

Exercise 6.26. What is the pushout of the functions $f : \underline{4} \to \underline{5}$ and $g : \underline{4} \to \underline{3}$ pictured below?



$f : \underline{4} \to \underline{5}$



$g : \underline{4} \to \underline{3}$

◊

Check your answer using the abstract description from Example 6.25.

Example 6.27. Suppose a category $\mathcal{C}$ has an initial object $\varnothing$. For any two objects $X, Y \in \operatorname{Ob} \mathcal{C}$, there is a unique morphism $f : \varnothing \to X$ and a unique morphism $g : \varnothing \to Y$; this is what it means for $\varnothing$ to be initial.

The diagram $X \xleftarrow{f} \varnothing \xrightarrow{g} Y$ has a pushout in $\mathcal{C}$ iff X and Y have a coproduct in $\mathcal{C}$, and the pushout and the coproduct will be the same. Indeed, suppose X and Y have a coproduct $X + Y$; then the diagram to the left

$$\begin{array}{ccc} \varnothing & \xrightarrow{f} & X \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle \iota_X} \\ Y & \xrightarrow[\iota_Y]{} & X + Y \end{array} \qquad \begin{array}{ccc} \varnothing & \xrightarrow{f} & X \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle x} \\ Y & \xrightarrow[y]{} & T \end{array}$$

commutes (why?[1]), and for any object T and commutative diagram as to the right, there is a unique map $X + Y \to T$ making the diagram as in Eq. (6.21) commute (why?[2]). This shows that $X + Y$ is a pushout, $X +_\varnothing Y \cong X + Y$.

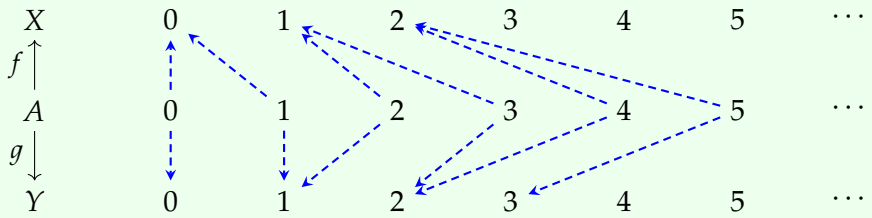
Similarly, if a pushout $X +_\varnothing Y$ exists, then it satisfies the universal property of the coproduct (why?[3]).

Exercise 6.28. In Example 6.27 we asked "why?" three times.

1. Give a justification for "why?[1]".
2. Give a justification for "why?[2]".
3. Give a justification for "why?[3]". ◊

Example 6.29. Let $A = X = Y = \mathbb{N}$. Consider the functions $f : A \to X$ and $g : A \to Y$

given by the 'floor' functions, $f(a) \coloneqq \lfloor a/2 \rfloor$ and $g(a) \coloneqq \lfloor (a+1)/2 \rfloor$.



What is their pushout? Let's figure it out using the definition.

If T is any other set and we have maps $x\colon X \to T$ and $y\colon Y \to T$ that commute with f and g, i.e. $f \mathbin{\fatsemi} x = g \mathbin{\fatsemi} y$, then this commutativity implies that

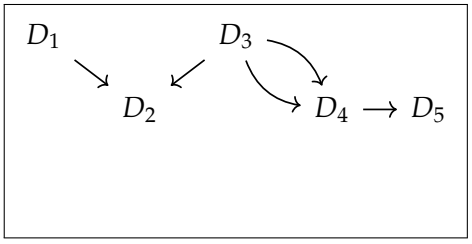
$$y(0) = y(g(0)) = x(f(0)) = x(0).$$

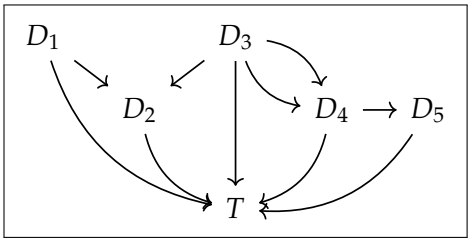
In other words, Y's 0 and X's 0 go to the same place in T, say t. But since $f(1) = 0$ and $g(1) = 1$, we also have that $t = x(0) = x(f(1)) = y(g(1)) = y(1)$. This means Y's 1 goes to t also. But since $g(2) = 1$ and $f(2) = 1$, we also have that $t = g(1) = y(g(2)) = x(f(2)) = x(1)$, which means that X's 1 also goes to t. One can keep repeating this and find that every element of Y and every element of X go to t! Using mathematical induction, one can prove that the pushout is in fact a 1-element set, $X \sqcup_A Y \cong \{1\}$.

6.2.4 Finite colimits

Initial objects, coproducts, and pushouts are all types of colimits. We gave the general definition of colimit in Section 3.5.4. Just as a limit in $\mathcal{C}$ is a terminal object in a category of cones over a diagram $D\colon \mathcal{J} \to \mathcal{C}$, a colimit is an initial object in a category of cocones over some diagram $D\colon \mathcal{J} \to \mathcal{C}$. For our purposes it is enough to discuss finite colimits—i.e. when $\mathcal{J}$ is a finite category—which subsume initial objects, coproducts, and pushouts.[4]

In Definition 3.102, cocones in $\mathcal{C}$ are defined to be cones in $\mathcal{C}^{\mathrm{op}}$. For visualization purposes, if $D\colon \mathcal{J} \to \mathcal{C}$ looks like the diagram to the left, then a cocone on it shown in the diagram to the right:





Here, any two parallel paths that end at T are equal in $\mathcal{C}$.

[4] If a category $\mathcal{J}$ has finitely many morphisms, we say that $\mathcal{J}$ is a *finite category*. Note that in this case it must have finitely many objects too, because each object $j \in \mathrm{Ob}\,\mathcal{J}$ has its own identity morphism id_j.

Definition 6.30. We say that a category $\mathcal{C}$ *has finite colimits* if a colimit, $\operatorname{colim}_{\mathcal{J}} D$, exists whenever $\mathcal{J}$ is a finite category and $D\colon \mathcal{J} \to \mathcal{C}$ is a diagram.

Example 6.31. The initial object in a category $\mathcal{C}$, if it exists, is the colimit of the functor $!\colon \mathbf{0} \to \mathcal{C}$, where $\mathbf{0}$ is the category with no objects and no morphisms, and $!$ is the unique such functor. Indeed, a cocone over $!$ is just an object of $\mathcal{C}$, and so the initial cocone over $!$ is just the initial object of $\mathcal{C}$.

Note that $\mathbf{0}$ has finitely many objects (none); thus initial objects are finite colimits.

We often want to know that a category $\mathcal{C}$ has *all* finite colimits (in which case, we often drop the 'all' and just say '$\mathcal{C}$ has finite colimits'). To check that $\mathcal{C}$ has (all) finite colimits, it's enough to check it has a few simpler forms of colimit, which generate all the rest.

Proposition 6.32. Let $\mathcal{C}$ be a category. The following are equivalent:

1. $\mathcal{C}$ has all finite colimits.
2. $\mathcal{C}$ has an initial object and all pushouts.
3. $\mathcal{C}$ has all coequalizers and all finite coproducts.

Proof. We will not give precise details here, but the key idea is an inductive one: one can build arbitrary finite diagrams using some basic building blocks. Full details can be found in [Bor94, Prop 2.8.2]. □

Example 6.33. Let $\mathcal{C}$ be a category with all pushouts, and suppose we want to take the colimit of the following diagram in $\mathcal{C}$:

$$\begin{array}{ccccc} & & B & \longrightarrow & Z \\ & & \downarrow & & \\ A & \longrightarrow & C & & \\ \downarrow & & & & \\ D & & & & \end{array} \tag{6.34}$$

In it we see two diagrams ready to be pushed out, and we know how to take pushouts. So suppose we do that; then we see another pushout diagram so we take the pushout again:

$$\begin{array}{ccccc} & & B & \longrightarrow & Z \\ & & \downarrow & \ulcorner & \downarrow \\ A & \longrightarrow & Y & \longrightarrow & R \\ \downarrow & \ulcorner & \downarrow & & \\ X & \longrightarrow & Q & & \end{array} \qquad \begin{array}{ccccc} & & B & \longrightarrow & Z \\ & & \downarrow & \ulcorner & \downarrow \\ A & \longrightarrow & Y & \longrightarrow & R \\ \downarrow & \ulcorner & \downarrow & \ulcorner & \downarrow \\ X & \longrightarrow & Q & \longrightarrow & S \end{array}$$

is the result—consisting of the object S, together with all the morphisms from the original diagram to S—the colimit of the original diagram? One can check that it indeed has the correct universal property and thus is a colimit.

Exercise 6.35. Check that the pushout of pushouts from Example 6.33 satisfies the universal property of the colimit for the original diagram, Eq. (6.34). ◊

We have already seen that the categories **FinSet** and **Set** both have an initial object and pushouts. We thus have the following corollary.

Corollary 6.36. The categories **FinSet** and **Set** have (all) finite colimits.

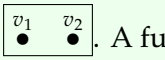
In Theorem 3.95 we gave a general formula for computing finite limits in **Set**. It is also possible to give a formula for computing finite colimits. There is a duality between products and coproducts and between subobjects and quotient objects, so whereas a finite limit is given by a subset of a product, a finite colimit is given by a quotient of a coproduct.

Theorem 6.37. Let $\mathcal{J}$ be presented by the finite graph (V, A, s, t) and some equations, and let $D\colon \mathcal{J} \to \mathbf{Set}$ be a diagram. Consider the set

$$\operatorname*{colim}_{\mathcal{J}} D := \big\{(v, d) \mid v \in V \text{ and } d \in D(v)\big\}/\sim$$

where this denotes the set of equivalence classes under the equivalence relation $\sim$ generated by putting $(v, d) \sim (w, e)$ if there is an arrow $a\colon v \to w$ in J such that $D(a)(d) = e$. Then this set, together with the functions $\iota_v\colon D(v) \to \operatorname{colim}_{\mathcal{J}} D$ given by sending $d \in D(v)$ to its equivalence class, constitutes a colimit of D.

Example 6.38. Recall that an initial object is the colimit on the empty graph. The formula thus says the initial object in **Set** is the empty set $\varnothing$: there are no $v \in V$.

Example 6.39. A coproduct is a colimit on the graph $\mathcal{J} =$ 

. A functor $D\colon \mathcal{J} \to \mathbf{Set}$ can be identified with a choice of two sets, $X := D(v_1)$ and $Y := D(v_2)$. Since there are no arrows in $\mathcal{J}$, the equivalence relation $\sim$ is vacuous, so the formula in Theorem 6.37 says that a coproduct is given by

$$\{(v, d) \mid d \in D(v), \text{ where } v = v_1 \text{ or } v = v_2\}.$$

In other words, the coproduct of sets X and Y is their disjoint union $X \sqcup Y$, as expected.

Example 6.40. If $\mathcal{J}$ is the category $\mathbf{1} = \boxed{\overset{v}{\bullet}}$, the formula in Theorem 6.37 yields the set

$$\{(v, d) \mid d \in D(v)\}$$

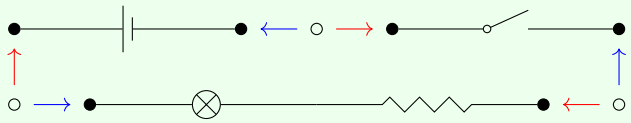
This is isomorphic to the set $D(v)$. In other words, if X is a set considered as a diagram $X \colon \mathbf{1} \to \mathbf{Set}$, then its colimit (like its limit) is just X again.

Exercise 6.41. Use the formula in Theorem 6.37 to show that pushouts—colimits on a diagram $X \xleftarrow{f} N \xrightarrow{g} Y$—agree with the description we gave in Example 6.25. ◊

Example 6.42. Another important type of finite colimit is the *coequalizer*. These are colimits over the graph $\boxed{\bullet \rightrightarrows \bullet}$ consisting of two parallel arrows.

Consider some diagram $X \overset{f}{\underset{g}{\rightrightarrows}} Y$ on this graph in **Set**. The coequalizer of this diagram is the set of equivalence classes of Y under equivalence relation generated by declaring $y \sim y'$ whenever there exists x in X such that $f(x) = y$ and $g(x) = y'$.

Let's return to the example circuit in the introduction to hint at why colimits are useful for interconnection. Consider the following picture:



We've redrawn this picture with one change: some of the arrows are now red, and others are now blue. If we let X be the set of white circles ∘, and Y be the set of black circles •, the blue and red arrows respectively define functions $f, g \colon X \to Y$. Let's leave the actual circuit components out of the picture for now; we're just interested in the dots. What is the coequalizer?

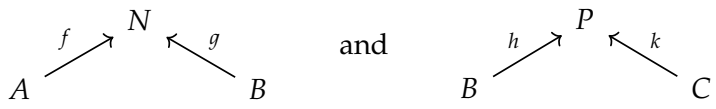
It is a three element set, consisting of one element for each newly-connected pair of •'s . Thus the colimit describes the set of terminals after performing the interconnection operation. In Section 6.4 we'll see how to keep track of the circuit components too.

6.2.5 Cospans

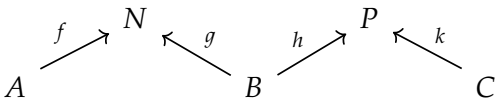
When a category $\mathcal{C}$ has finite colimits, an extremely useful way to package them is by considering the category of cospans in $\mathcal{C}$.

Definition 6.43. Let $\mathcal{C}$ be a category. A *cospan* in $\mathcal{C}$ is just a pair of morphisms to a common object $A \to N \leftarrow B$. The common object N is called the *apex* of the cospan and the other two objects A and B are called its *feet*.

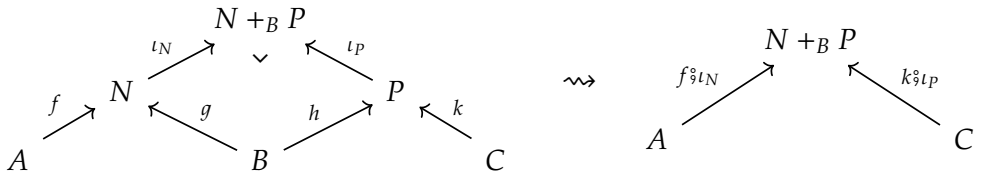
If we want to say that cospans form a category, we should begin by saying how composition would work. So suppose we have two cospans in $\mathcal{C}$



Since the right foot of the first is equal to the left foot of the second, we might stick them together into a diagram like this:



Then, if a pushout of $N \xleftarrow{g} B \xrightarrow{h} P$ exists in $\mathcal{C}$, as shown on the left, we can extract a new cospan in $\mathcal{C}$, as shown on the right:



(6.44)

It might look like we have achieved our goal, but we're missing some things. First, we need an identity on every object $C \in \mathrm{Ob}\,\mathcal{C}$; but that's not hard: use $C \to C \leftarrow C$ where both maps are identities in $\mathcal{C}$. More importantly, we don't know that $\mathcal{C}$ has all pushouts, so we don't know that every two sequential morphisms $A \to B \to C$ can be composed. And beyond that, there is a technical condition that when we form pushouts, we only get an answer 'up to isomorphism': anything isomorphic to a pushout counts as a pushout (check the definition to see why). We want all these different choices to count as the same thing, so we define two cospans to be equivalent iff there is an isomorphism between their respective apexes. That is, the cospan $A \to P \leftarrow B$ and $A \to P' \leftarrow B$ in the diagram shown left below are equivalent iff there is an isomorphism $P \cong P'$ making the diagram to the right commute:

Now we are getting somewhere. As long as our category $\mathcal{C}$ has pushouts, we are in business: $\mathbf{Cospan}_{\mathcal{C}}$ will form a category. But in fact, we are very close to getting more. If we also demand that $\mathcal{C}$ has an initial object $\varnothing$ as well, then we can upgrade $\mathbf{Cospan}_{\mathcal{C}}$ to a symmetric monoidal category.

Recall from Proposition 6.32 that a category $\mathcal{C}$ has all finite colimits iff it has an initial object and all pushouts.

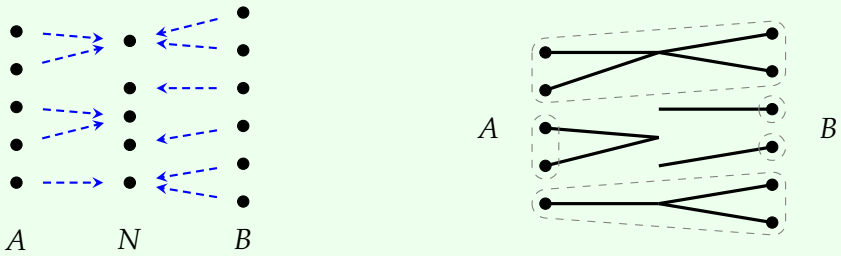
Definition 6.45. Let $\mathcal{C}$ be a category with finite colimits. Then there exists a category $\mathbf{Cospan}_{\mathcal{C}}$ with the same objects as $\mathcal{C}$, i.e. $\mathrm{Ob}(\mathbf{Cospan}_{\mathcal{C}}) = \mathrm{Ob}(\mathcal{C})$, where the morphisms $A \to B$ are the (equivalence classes of) cospans from A to B, and composition is given by the above pushout construction.

There is a symmetric monoidal structure on this category, denoted $(\mathbf{Cospan}_{\mathcal{C}}, \varnothing, +)$. The monoidal unit is the initial object $\varnothing \in \mathcal{C}$ and the monoidal product is given by coproduct. The coherence isomorphisms, e.g. $A + \varnothing \cong A$, can be defined in a similar way to those in Exercise 6.18.

It is a straightforward but time-consuming exercise to verify that $(\mathbf{Cospan}_{\mathcal{C}}, \varnothing, +)$ from Definition 6.45 really does satisfy all the axioms of a symmetric monoidal category, but it does.

Example 6.46. The category **FinSet** has finite colimits (see 6.36). So, we can define a symmetric monoidal category $\mathbf{Cospan}_{\mathbf{FinSet}}$. What does it look like? It looks a lot like wires connecting ports.

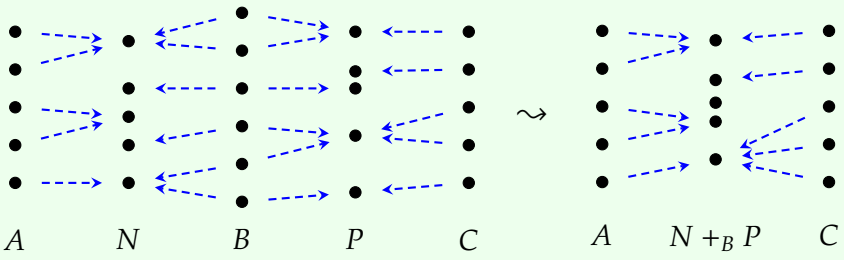
The objects of $\mathbf{Cospan}_{\mathbf{FinSet}}$ are finite sets; here let's draw them as collections of $\bullet$'s. The morphisms are cospans of functions. Let A and N be five element sets, and B be a six element set. Below are two depictions of a cospan $A \xrightarrow{f} N \xleftarrow{g} B$.



In the depiction on the left, we simply represent the functions f and g by drawing arrows from each $a \in A$ to $f(a)$ and each $b \in B$ to $g(b)$. In the depiction on the right, we make this picture resemble wires a bit more, simply drawing a wire where before we had an arrow, and removing the unnecessary center dots. We also draw a dotted line around points that are connected, to emphasize an important perspective, that cospans establish that certain ports are connected, i.e. part of the same equivalence class.

The monoidal category $\mathbf{Cospan}_{\mathbf{FinSet}}$ then provides two operations for combining cospans: composition and monoidal product. Composition is given by taking the pushout of the maps coming from the common foot, as described in Definition 6.45. Here is an example of cospan composition, where all the functions are depicted with

arrow notation:

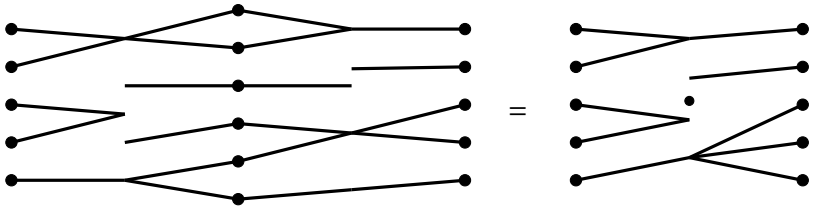


(6.47)

The monoidal product is given simply by the disjoint union of two cospans; in pictures it is simply combining two cospans by stacking one above another.

Exercise 6.48. In Eq. (6.47) we showed morphisms $A \to B$ and $B \to C$ in $\mathbf{Cospan}_{\mathbf{FinSet}}$. Draw their monoidal product as a morphism $A + B \to B + C$ in $\mathbf{Cospan}_{\mathbf{FinSet}}$. ◊

Exercise 6.49. Depicting the composite of cospans in Eq. (6.47) with the wire notation gives



(6.50)

Comparing Eq. (6.47) and Eq. (6.50), describe the composition rule in $\mathbf{Cospan}_{\mathbf{FinSet}}$ in terms of wires and connected components. ◊

6.3 Hypergraph categories

A hypergraph category is a type of symmetric monoidal category whose wiring diagrams are networks. We will soon see that electric circuits can be organized into a hypergraph category; this is what we've been building up to. But to define hypergraph categories, it is useful to first introduce Frobenius monoids.

6.3.1 Frobenius monoids

The pictures of cospans we saw above, e.g. in Eq. (6.50) look something like icons in signal flow graphs (see Section 5.3.2): various wires merge and split, initialize and terminate. And these follow the same rules they did for linear relations, which we briefly discussed in Exercise 5.84. There's a lot of potential for confusion, so let's start from scratch and build back up.

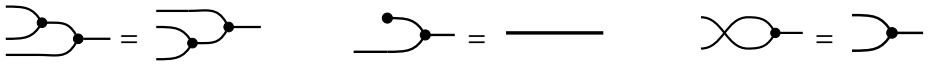
In any symmetric monoidal category $(\mathcal{C}, I, \otimes)$, recall from Section 4.4.2 that objects can be drawn as wires and morphisms can be drawn as boxes. Particularly noteworthy morphisms might be iconified as dots rather than boxes, to indicate that the morphisms

there are not arbitrary but notation-worthy. One case of this is when there is an object X with special "abilities", e.g. the ability to duplicate into two, or disappear into nothing.

To make this precise, recall from Definition 5.65 that a commutative monoid (X, μ, η) in symmetric monoidal category $(\mathcal{C}, I, \otimes)$ is an object X of $\mathcal{C}$ together with (noteworthy) morphisms

$\mu\colon X \otimes X \to X$ $\qquad$ $\eta\colon I \to X$

obeying



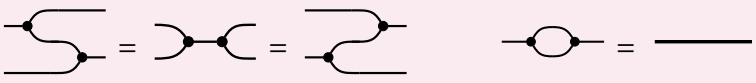
(associativity) (unitality) (commutativity) (6.51)

where $\times$ is the symmetry on $X \otimes X$. A cocommutative cocomonoid (X, δ, ϵ) is an object X with maps $\delta\colon X \to X \otimes X$, $\epsilon\colon X \to I$, obeying the mirror images of the laws in Eq. (6.51).

Suppose X has both the structure of a commutative monoid and cocommutative comonoid, and consider a wiring diagram built only from the icons μ, η, δ, and ϵ, where every wire is labeled X. These diagrams have a left and right, and are pictures of how ports on the left are connected to ports on the right. The commutative monoid and cocommutative comonoid axioms thus both express when to consider two such connection pictures should be considered the same. For example, associativity says the order of connecting ports on the left doesn't matter; coassociativity (not drawn) says the same for the right.

If you want to go all the way and say "all I care about is which port is connected to which; I don't even care about left and right", then you need a few more axioms to say how the morphisms μ and δ, the merger and the splitter, interact.

Definition 6.52. Let X be an object in a symmetric monoidal category $(\mathcal{C}, \otimes, I)$. A *Frobenius structure* on X consists of a 4-tuple $(\mu, \eta, \delta, \epsilon)$ such that (X, μ, η) is a commutative monoid and (X, δ, ϵ) is a cocommutative comonoid, which satisfies the six equations above ((co-)associativity, (co-)unitality, (co-)commutativity), as well as the following three equations:

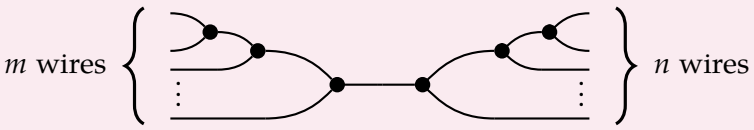


(the Frobenius law) (the special law) (6.53)

We refer to an object X equipped with a Frobenius structure as a *special commutative Frobenius monoid*, or just *Frobenius monoid* for short.

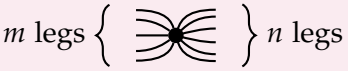
With these two equations, it turns out that two morphisms $X^{\otimes m} \to X^{\otimes n}$—defined by composing and tensoring identities on X and the noteworthy morphisms μ, δ, etc.—are equal if and only if their string diagrams connect the same ports. This link between connectivity, and Frobenius monoids can be made precise as follows.

Definition 6.54. Let $(X, \mu, \eta, \delta, \epsilon)$ be a Frobenius monoid in a monoidal category $(\mathcal{C}, I, \otimes)$. Let $m, n \in \mathbb{N}$. Define $s_{m,n} \colon X^{\otimes m} \to X^{\otimes n}$ to be the following morphism



It can be written formally as $(m-1)$ μ's followed by $(n-1)$ δ's, with special cases when $m = 0$ or $n = 0$.

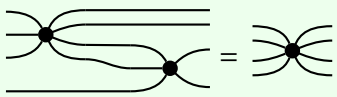
We call $s_{m,n}$ the *spider of type* (m, n), and can draw it more simply as the icon



So a special commutative Frobenius monoid, aside from being a mouthful, is a 'spiderable' wire. You agree that in any monoidal category wiring diagram language, wires represent objects and boxes represent morphisms? Well in our weird way of talking, if a wire is spiderable, it means that we have a bunch of morphisms $\mu, \eta, \delta, \epsilon, \sigma$ that we can combine without worrying about the order of doing so: the result is just "how many in's, and how many out's": a spider. Here's a formal statement.

Theorem 6.55. Let $(X, \mu, \eta, \delta, \epsilon)$ be a Frobenius monoid in a monoidal category $(\mathcal{C}, I, \otimes)$. Suppose that we have a map $f \colon X^{\otimes m} \to X^{\otimes n}$ each constructed from spiders and the symmetry map $\sigma \colon X^{\otimes 2} \to X^{\otimes 2}$ using composition and the monoidal product, and such that the string diagram of f has only one connected component. Then it is a spider: $f = s_{m,n}$.

Example 6.56. As the following two morphisms both (i) have the same number of inputs and outputs, (ii) are constructed only from spiders, and (iii) are connected, Theorem 6.55 immediately implies they are equal:

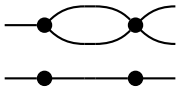


Exercise 6.57. Let X be an object equipped with a Frobenius structure. Which of the morphisms $X \otimes X \to X \otimes X \otimes X$ in the following list are necessarily equal?

1.

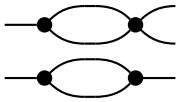
2.

3.

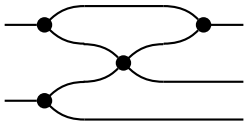


4.

5.



6.



◊

Back to cospans. Another way of understanding Frobenius monoids is to relate them to cospans. Recall the notion of prop presentation from Definition 5.33.

Theorem 6.58. Consider the four-element set $G := \{\mu, \eta, \delta, \epsilon\}$ and define $in, out \colon G \to \mathbb{N}$ as follows:

$$\begin{aligned} in(\mu) &:= 2, & in(\eta) &:= 0, & in(\delta) &:= 1, & in(\epsilon) &:= 1, \\ out(\mu) &:= 1, & out(\eta) &:= 1, & out(\delta) &:= 2, & out(\epsilon) &:= 0. \end{aligned}$$

Let E be the set of Frobenius axioms, i.e. the nine equations from Definition 6.52. Then the free prop on (G, E) is equivalent, as a symmetric monoidal category,[a] to $\mathbf{Cospan}_{\mathbf{FinSet}}$.

[a] We will not explain precisely what it means to be equivalent as a symmetric monoidal category, but you probably have some idea: "they are the same for all category-theoretic intents and purposes." The idea is similar to that of equivalence of categories, as explained in Remark 3.59.

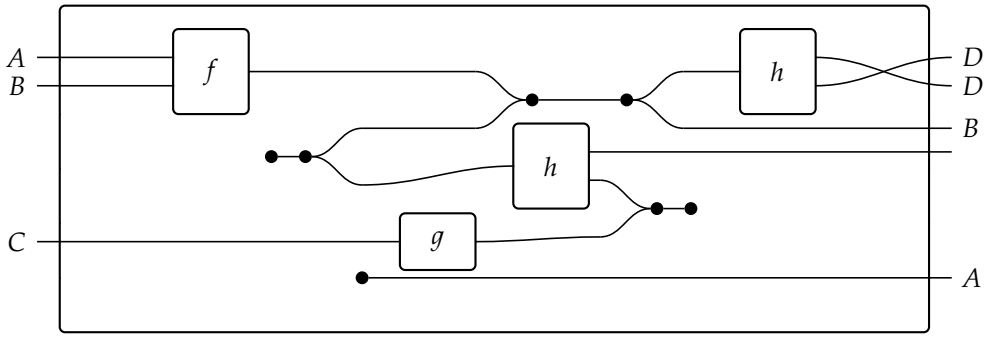
Thus we see that ideal wires, connectivity, cospans, and objects with Frobenius structures are all intimately related. We use Frobenius structures (all that splitting, merging, initializing, and terminating stuff) as a way to capture the grammar of circuit diagrams.

6.3.2 Wiring diagrams for hypergraph categories

We introduce hypergraph categories through their wiring diagrams. Just like for monoidal categories, the formal definition is just the structure required to unambiguously interpret these diagrams.

Indeed, our interest in hypergraph categories is best seen in their wiring diagrams. The key idea is that wiring diagrams for hypergraph categories are network diagrams. This means, in addition to drawing labeled boxes with inputs and outputs, as we can for monoidal categories, and in addition to bending these wires around as we can for compact closed categories, we are allowed to split, join, terminate, and initialize wires.

Here is an example of a wiring diagram that represents a composite of morphisms in a hypergraph category



We have suppressed some of the object/wire labels for readability, since all types can be inferred from the labeled ones.

Exercise 6.59.

1. What label should be on the input to h?
2. What label should be on the output of g?
3. What label should be on the fourth output wire of the composite? ◊

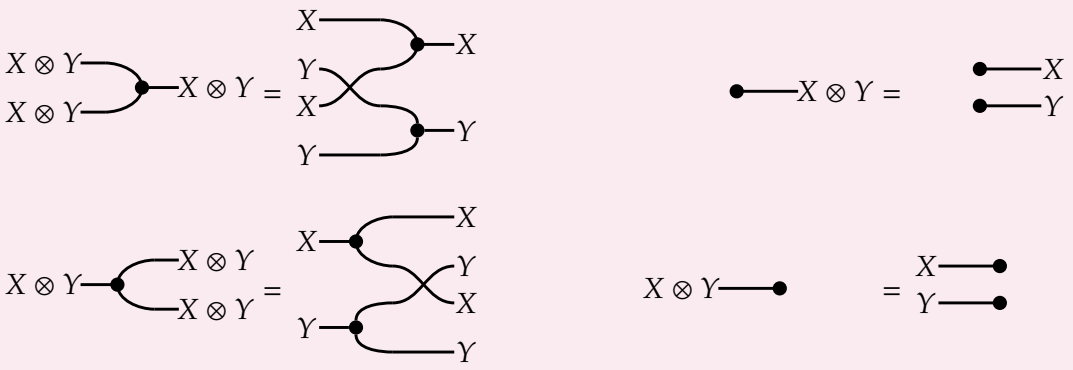
Thus hypergraph categories are general enough to talk about all network-style diagrammatic languages, like circuit diagrams.

6.3.3 Definition of hypergraph category

We are now ready to define hypergraph categories formally. Since the wiring diagrams for hypergraph categories are just those for symmetric monoidal categories with a few additional icons, the definition is relatively straightforward: we just want a Frobenius structure on every object. The only coherence condition is that these interact nicely with the monoidal product.

Definition 6.60. A *hypergraph category* is a symmetric monoidal category $(\mathcal{C}, I, \otimes)$ in which each object X is equipped with a Frobenius structure $(X, \mu_X, \delta_X, \eta_X, \epsilon_X)$ such

that



for all objects X, Y, and such that $\eta_I = \mathrm{id}_I = \epsilon_I$.

A *hypergraph prop* is a hypergraph category that is also a prop, e.g. $\mathrm{Ob}(\mathcal{C}) = \mathbb{N}$, etc.

Example 6.61. For any $\mathcal{C}$ with finite colimits, $\mathbf{Cospan}_{\mathcal{C}}$ is a hypergraph category. The Frobenius morphisms $\mu_X, \delta_X, \eta_X, \epsilon_X$ for each object X are constructed using the universal properties of colimits:

$$\begin{aligned}
\mu_X &:= \left(X + X \xrightarrow{[\mathrm{id}_X, \mathrm{id}_X]} X \xleftarrow{\mathrm{id}_X} X\right) \\
\eta_X &:= \left(\varnothing \xrightarrow{!_X} X \xleftarrow{\mathrm{id}_X} X\right) \\
\delta_X &:= \left(X \xrightarrow{\mathrm{id}_X} X \xleftarrow{[\mathrm{id}_X, \mathrm{id}_X]} X + X\right) \\
\epsilon_X &:= \left(X \xrightarrow{\mathrm{id}_X} X \xleftarrow{!_X} \varnothing\right)
\end{aligned}$$

Exercise 6.62. By Example 6.61, the category $\mathbf{Cospan}_{\mathbf{FinSet}}$ is a hypergraph category. (In fact, it is equivalent to a hypergraph prop.) Draw the Frobenius morphisms for the object $\underline{1}$ in $\mathbf{Cospan}_{\mathbf{FinSet}}$ using both the function and wiring depictions as in Example 6.46. ◊

Exercise 6.63. Using your knowledge of colimits, show that the maps defined in Example 6.61 do indeed obey the special law (see Definition 6.52). ◊

Example 6.64. Recall the monoidal category $(\mathbf{Corel}, \varnothing, \sqcup)$ from Example 4.61; its objects are finite sets and its morphisms are corelations. Given a finite set X, define the corelation $\mu_X \colon X \sqcup X \to X$ such that two elements of $X \sqcup X \sqcup X$ are equivalent if and only if they come from the same underlying element of X. Define $\delta_X \colon X \to X \sqcup X$ in the same way, and define $\eta_X \colon \varnothing \to X$ and $\epsilon_X \colon X \to \varnothing$ such that no two elements of $X = \varnothing \sqcup X = X \sqcup \varnothing$ are equivalent.

These maps define a special commutative Frobenius monoid $(X, \mu_X, \eta_X, \delta_X, \epsilon_X)$,

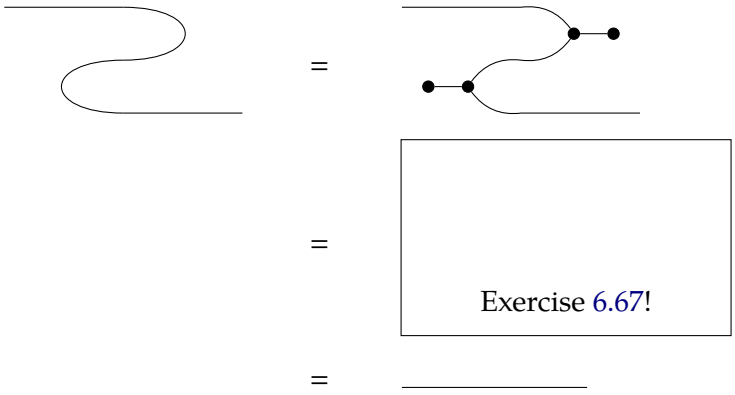
and in fact give **Corel** the structure of a hypergraph category.

Example 6.65. The prop of linear relations, which we briefly mentioned in Exercise 5.84, is a hypergraph category. In fact, it is a hypergraph category in two ways, by choosing either the black 'copy' and 'discard' generators or the white 'add' and 'zero' generators as the Frobenius maps.

We can generalize the construction we gave in Theorem 5.87.

Proposition 6.66. Hypergraph categories are self-dual compact closed categories, if we define the cup and cap to be

Proof. The proof is a straightforward application of the Frobenius and unitality axioms:



□

Exercise 6.67. Fill in the missing diagram in the proof of Proposition 6.66 using the equations from Eq. (6.51), their opposites, and Eq. (6.53). ◊

6.4 Decorated cospans

The goal of this section is to show how we can construct a hypergraph category whose morphisms are electric circuits. To do this, we first must introduce the notion of structure-preserving map for symmetric monoidal categories, a generalization of monoidal monotones known as symmetric monoidal functors. Then we introduce a general method—that of decorated cospans—for producing hypergraph categories. Doing all this will tie up lots of loose ends: colimits, cospans, circuits, and hypergraph categories.

6.4.1 Symmetric monoidal functors

Rough Definition 6.68. Let $(\mathcal{C}, I_\mathcal{C}, \otimes_\mathcal{C})$ and $(\mathcal{D}, I_\mathcal{D}, \otimes_\mathcal{D})$ be symmetric monoidal categories. To specify a *symmetric monoidal functor* (F, φ) between them,

(i) one specifies a functor $F\colon \mathcal{C} \to \mathcal{D}$;

(ii) one specifies a morphism $\varphi_I\colon I_\mathcal{D} \to F(I_\mathcal{C})$.

(iii) for each $c_1, c_2 \in \mathrm{Ob}(\mathcal{C})$, one specifies a morphism

$$\varphi_{c_1,c_2}\colon F(c_1) \otimes_\mathcal{D} F(c_2) \to F(c_1 \otimes_\mathcal{C} c_2),$$

natural in c_1 and c_2.

We call the various maps φ *coherence maps*. We require the coherence maps to obey bookkeeping axioms that ensure they are well behaved with respect to the symmetric monoidal structures on $\mathcal{C}$ and $\mathcal{D}$. If φ_I and φ_{c_1,c_2} are isomorphisms for all c_1, c_2, we say that (F, φ) is *strong*.

Example 6.69. Consider the power set functor $\mathsf{P}\colon \mathbf{Set} \to \mathbf{Set}$. It acts on objects by sending a set $S \in \mathbf{Set}$ to its set of subsets $\mathsf{P}(S) := \{R \subseteq S\}$. It acts on morphisms by sending a function $f\colon S \to T$ to the image map $\mathrm{im}_f\colon \mathsf{P}(S) \to \mathsf{P}(T)$, which maps $R \subseteq S$ to $\{f(r) \mid r \in R\} \subseteq T$.

Now consider the symmetric monoidal structure $(\{1\}, \times)$ on $\mathbf{Set}$ from Example 4.49. To make P a symmetric monoidal functor, we need to specify a function $\varphi_I\colon \{1\} \to \mathsf{P}(\{1\})$ and for all sets S and T, a functor $\varphi_{S,T}\colon \mathsf{P}(S) \times \mathsf{P}(T) \to \mathsf{P}(S \times T)$. One possibility is to define $\varphi_I(1)$ to be the maximal subset $\{1\} \subseteq \{1\}$, and given subsets $A \subseteq S$ and $B \subseteq T$, to define $\varphi_{S,T}(A, B)$ to be the product subset $A \times B \subseteq S \times T$. With these definitions, (P, φ) is a symmetric monoidal functor.

Exercise 6.70. Check that the maps $\varphi_{S,T}$ defined in Example 6.69 are natural in S and T. In other words, given $f\colon S \to S'$ and $g\colon T \to T'$, show that the diagram below commutes:

$$\begin{array}{ccc} \mathsf{P}(S) \times \mathsf{P}(T) & \xrightarrow{\varphi_{S,T}} & \mathsf{P}(S \times T) \\ {\scriptstyle \mathrm{im}_f \times \mathrm{im}_g}\downarrow & & \downarrow{\scriptstyle \mathrm{im}_{f \times g}} \\ \mathsf{P}(S') \times \mathsf{P}(T') & \xrightarrow[\varphi_{S',T'}]{} & \mathsf{P}(S' \times T') \end{array}$$

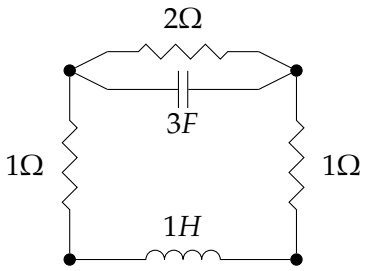
◊

6.4.2 Decorated cospans

Now that we have briefly introduced symmetric monoidal functors, we return to the task at hand: constructing a hypergraph category of circuits. To do so, we introduce the method of decorated cospans.

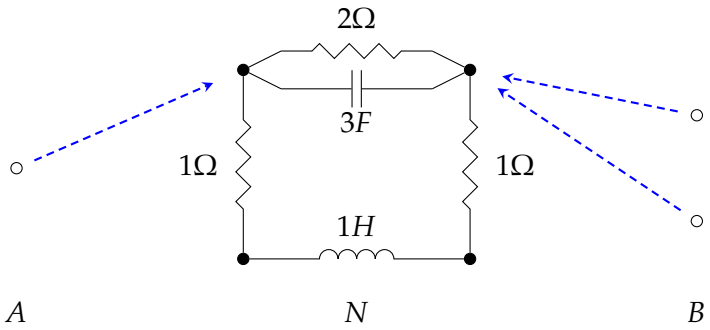
Circuits have lots of internal structure, but they also have some external ports—also called 'terminals'—by which to interconnect them with others. Decorated cospans are ways of discussing exactly that: things with external ports and internal structure.

To see how this works, let us start with the following example circuit:



(6.71)

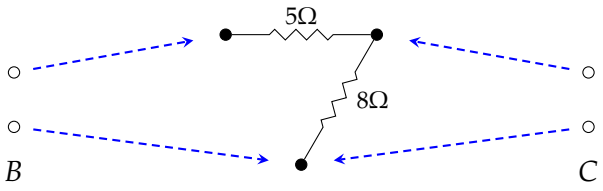
We might formally consider this as a graph on the set of four ports, where each edge is labeled by a type of circuit component (for example, the top edge would be labeled as a resistor of resistance 2Ω). For this circuit to be a morphism in some category, i.e. in order to allow for interconnection, we must equip the circuit with some notion of interface. We do this by marking the ports in the interface using functions from finite sets:



(6.72)

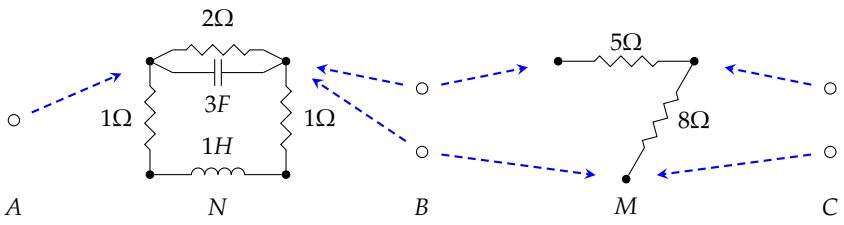
Let N be the set of nodes of the circuit. Here the finite sets A, B, and N are sets consisting of one, two, and four elements respectively, drawn as points, and the values of the functions $A \to N$ and $B \to N$ are indicated by the grey arrows. This forms a cospan in the category of finite sets, for which the apex set N has been *decorated* by our given circuit.

Suppose given another such decorated cospan with input B



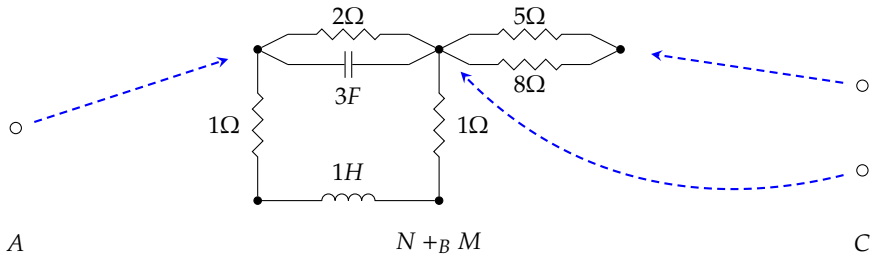
Since the output of the first equals the input of the second (both are B), we can stick

them together into a single diagram:



(6.73)

The composition is given by gluing the circuits along the identifications specified by B, resulting in the decorated cospan



(6.74)

We've seen this sort of gluing before when we defined composition of cospans in Definition 6.45. But now there's this whole 'decoration' thing; our goal is to formalize it.

Definition 6.75. Let $\mathcal{C}$ be a category with finite colimits, and $(F, \varphi)\colon (\mathcal{C}, +) \longrightarrow (\mathbf{Set}, \times)$ be a symmetric monoidal functor. An *F-decorated cospan* is a pair consisting of a cospan $A \xrightarrow{i} N \xleftarrow{o} B$ in $\mathcal{C}$ together with an element $s \in F(N)$.[5] We call (F, φ) the *decoration functor* and s the *decoration*.

The intuition here is to use $\mathcal{C} = \mathbf{FinSet}$, and, for each object $N \in \mathbf{FinSet}$, the functor F assigns the set of all legal decorations on a set N of nodes. When you choose an F-decorated cospan, you choose a set A of left-hand external ports, a set B of right-hand external ports, each of which maps to a set N of nodes, and you choose one of the available decorations on N nodes, taken from the set $F(N)$.

So, in our electrical circuit case, the decoration functor F sends a finite set N to the set of circuit diagrams—graphs whose edges are labeled by resistors, capacitors, etc.—that have N vertices.

Our goal is still to be able to compose such diagrams; so how does that work exactly? Basically one combines the way cospans are composed with the structures defining our decoration functor: namely F and φ.

Let $(A \xrightarrow{f} N \xleftarrow{g} B, s)$ and $(B \xrightarrow{h} P \xleftarrow{k} C, t)$ represent decorated cospans. Their composite is represented by the composite of the cospans $A \xrightarrow{f} N \xleftarrow{g} B$ and $B \xrightarrow{h} P \xleftarrow{k} C$,

[5] Just like in Definition 6.45, we should technically use equivalence classes of cospans. We will elide this point to get the bigger idea across. The interested reader should consult Section 6.6.

paired with the following element of $F(N +_B P)$:

$$F([\iota_N, \iota_P])(\varphi_{N,P}(s,t)) \tag{6.76}$$

That's rather compact! We'll unpack it, in a concrete case, in just a second. But let's record a theorem first.

Theorem 6.77. Given a category $\mathcal{C}$ with finite colimits and a symmetric monoidal functor $(F, \varphi)\colon (\mathcal{C}, +) \longrightarrow (\mathbf{Set}, \times)$, there is a hypergraph category $\mathbf{Cospan}_F$ whose objects are the objects of $\mathcal{C}$, and whose morphisms are equivalence classes of F-decorated cospans.

The symmetric monoidal and hypergraph structures are derived from those on $\mathbf{Cospan}_{\mathcal{C}}$.

Exercise 6.78. Suppose you're worried that the notation $\mathbf{Cospan}_{\mathcal{C}}$ looks like the notation $\mathbf{Cospan}_F$, even though they're very different. An expert tells you "they're not so different; one is a special case of the other. Just use the constant functor $F(c) \coloneqq \{*\}$." What does the expert mean? ◊

6.4.3 Electric circuits

In order to work with the above abstractions, we will get a bit more precise about the circuits example and then have a detailed look at how composition works in decorated cospan categories.

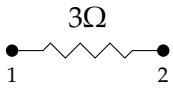
Let's build some circuits. To begin, we'll need to choose which components we want in our circuit. This is simply a matter of what's in our electrical toolbox. Let's say we're carrying some lightbulbs, switches, batteries, and resistors of every possible resistance. That is, define a set

$$C \coloneqq \{\texttt{light}, \texttt{switch}, \texttt{battery}\} \sqcup \{x\Omega \mid x \in \mathbb{R}^+\}.$$

To be clear, the Ω are just labels; the above set is isomorphic to $\{\texttt{light}, \texttt{switch}, \texttt{battery}\} \sqcup \mathbb{R}^+$. But we write C this way to remind us that it consists of circuit components. If we wanted, we could also add inductors, capacitors, and even elements connecting more than two ports, like transistors, but let's keep things simple for now.

Given our set C, a C-circuit is just a graph (V, A, s, t), where $s, t\colon A \to V$ are the source and target functions, together with a function $\ell\colon A \to C$ labeling each edge with a certain circuit component from C.

For example, we might have the simple case of $V = \{1, 2\}$, $A = \{e\}$, $s(e) = 1$, $t(e) = 2$—so e is an edge from 1 to 2—and $\ell(e) = 3\Omega$. This represents a resistor with resistance 3Ω:



Note that in the formalism we have chosen, we have multiple ways to represent any circuit, as our representations explicitly choose directions for the edges. The above resistor could also be represented by the 'reversed graph', with data $V = \{1, 2\}$, $A = \{e\}$, $s(e) = 2$, $t(e) = 1$, and $\ell(e) = 3F$.

Exercise 6.79. Write a tuple (V, A, s, t, ℓ) that represents the circuit in Eq. (6.71). ◊

A decoration functor for circuits. We want C-circuits to be our decorations, so let's use them to define a decoration functor as in Definition 6.75. We'll call the functor (Circ, ψ). We start by defining the functor part

$$\mathsf{Circ}\colon (\mathbf{FinSet}, +) \longrightarrow (\mathbf{Set}, \times)$$

as follows. On objects, simply send a finite set V to the set of C-circuits:

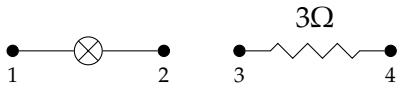
$$\mathsf{Circ}(V) := \{(V, A, s, t, \ell) \mid \text{ where } s, t\colon A \to V, \ell\colon E \to C\}.$$

On morphisms, Circ sends a function $f\colon V \to V'$ to the function

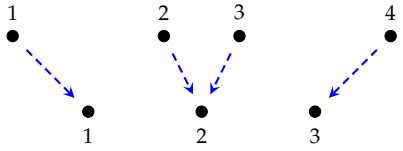
$$\begin{aligned} \mathsf{Circ}(f)\colon \mathsf{Circ}(V) &\longrightarrow \mathsf{Circ}(V'); \\ (V, A, s, t, \ell) &\longmapsto \big(V', A, (s \fatsemi f), (t \fatsemi f), \ell\big). \end{aligned}$$

This defines a functor; let's explore it a bit in an exercise.

Exercise 6.80. To understand this functor better, let $c \in \mathsf{Circ}(\underline{4})$ be the circuit



and let $f\colon \underline{4} \to \underline{3}$ be the function



Draw a picture of the circuit $\mathsf{Circ}(f)(c)$. ◊

We're trying to get a decoration functor (Circ, ψ) and so far we have Circ. For the coherence maps $\psi_{V,V'}$ for finite sets V, V', we define

$$\begin{aligned} \psi_{V,V'}\colon \mathsf{Circ}(V) \times \mathsf{Circ}(V') &\longrightarrow \mathsf{Circ}(V + V'); \\ \big((V, A, s, t, \ell), (V', A', s', t', \ell')\big) &\longmapsto (V + V', A + A', s + s', t + t', [\ell, \ell']). \end{aligned} \tag{6.81}$$

This is simpler than it may look: it takes a circuit on V and a circuit on V', and just considers them together as a circuit on the disjoint union of vertices $V + V'$.

Exercise 6.82. Suppose we have circuits

in $\mathsf{Circ}(\underline{2})$. Use the definition of $\psi_{V,V'}$ from (6.81) to figure out what 4-vertex circuit $\psi_{\underline{2},\underline{2}}(b, s) \in \mathsf{Circ}(\underline{2} + \underline{2}) = \mathsf{Circ}(\underline{4})$ should be, and draw a picture. ◊

Open circuits using decorated cospans. From the above data, just a monoidal functor $(\mathsf{Circ}, \psi)\colon (\mathbf{FinSet}, +) \to (\mathbf{Set}, \times)$, we can construct our promised hypergraph category of circuits!

Our notation for this category is $\mathbf{Cospan}_{\mathsf{Circ}}$. Following Theorem 6.77, the objects of this category are the same as the objects of **FinSet**, just finite sets. We'll reprise our notation from the introduction and Example 6.42, and draw these finite sets as collections of white circles ∘. For example, we'll represent the object $\underline{2}$ of $\mathbf{Cospan}_{\mathsf{Circ}}$ as two white circles:

∘ ∘

These white circles mark interface points of an open circuit.

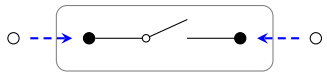
More interesting than the objects, however, are the morphisms in $\mathbf{Cospan}_{\mathsf{Circ}}$. These are open circuits. By Theorem 6.77, a morphism $\underline{m} \to \underline{n}$ is a Circ-decorated cospan: that is, cospan $\underline{m} \to \underline{p} \leftarrow \underline{n}$ together with an element c of $\mathsf{Circ}(\underline{p})$. As an example, consider the cospan $\underline{1} \xrightarrow{i_1} \underline{2} \xleftarrow{i_2} \underline{1}$ where $i_1(1) = 1$ and $i_2(1) = 2$, equipped with the battery element of $\mathsf{Circ}(\underline{2})$ connecting node 1 and node 2. We'll depict this as follows:

(6.83)

Exercise 6.84. Morphisms of $\mathbf{Cospan}_{\mathsf{Circ}}$ are Circ-decorated cospans, as defined in Definition 6.75. This means (6.83) depicts a cospan together with a *decoration*, which is some C-circuit $(V, A, s, t, \ell) \in \mathsf{Circ}(\underline{2})$. What is it? ◊

Let's now see how the hypergraph operations in $\mathbf{Cospan}_{\mathsf{Circ}}$ can be used to construct electric circuits.

Composition in $\mathbf{Cospan}_{\mathsf{Circ}}$. First we'll consider composition. Consider the following decorated cospan from $\underline{1}$ to $\underline{1}$:

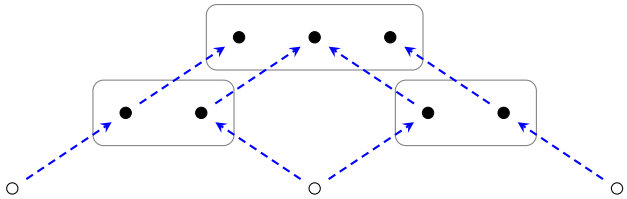


Since this and the circuit in (6.83) are both morphisms $\underline{1} \to \underline{1}$, we may compose them to get another morphism $\underline{1} \to \underline{1}$. How do we do this? There are two parts: to get the new cospan, we simply compose the cospans of our two circuits, and to get the new decoration, we use the formula $\mathsf{Circ}([\iota_N, \iota_P])(\psi_{N,P}(s, t))$ from (6.76). Again, this is rather compact! Let's unpack it together.

We'll start with the cospans. The cospans we wish to compose are

 and 

(We simply ignore the decorations for now.) If we pushout over the common set $\underline{1} = \{\circ\}$, we obtain the pushout square



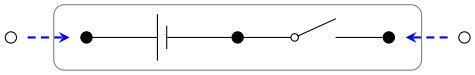
(6.85)

This means the composite cospan is

In the meantime, we already had you start us off unpacking the formula for the new decoration. You told us what the map $\psi_{\underline{2},\underline{2}}$ does in Exercise 6.82. It takes the two decorations, both circuits in $\mathsf{Circ}(\underline{2})$, and turns them into the single, disjoint circuit

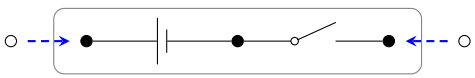
in $\mathsf{Circ}(\underline{4})$. So this is what the $\psi_{N,P}(s,t)$ part means. What does the $[\iota_N, \iota_P]$ mean? Recall this is the copairing of the pushout maps, as described in Examples 6.14 and 6.25. In our case, the relevant pushout square is given by (6.85), and $[\iota_N, \iota_P]$ is in fact the function f from Exercise 6.80! This means the decoration on the composite cospan is

Putting this all together, the composite circuit is

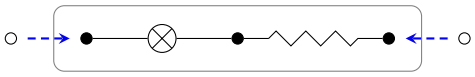


Exercise 6.86. Refer back to the example at the beginning of Section 6.4.2. In particular, consider the composition of circuits in Eq. (6.73). Express the two circuits in this diagram as morphisms in $\mathbf{Cospan}_{\mathsf{Circ}}$, and compute their composite. Does it match the picture in Eq. (6.74)? ◊

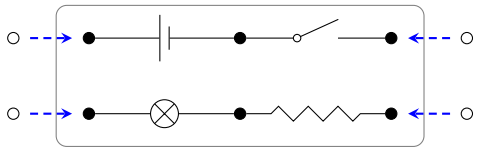
Monoidal products in $\mathbf{Cospan}_{\mathsf{Circ}}$. Monoidal products in $\mathbf{Cospan}_{\mathsf{Circ}}$ are much simpler than composition. On objects, we again just work as in **FinSet**: we take the disjoint union of finite sets. Morphisms again have a cospan, and a decoration. For cospans, we again just work in $\mathbf{Cospan}_{\mathbf{FinSet}}$: given two cospans $A \to M \leftarrow B$ and $C \to N \leftarrow D$, we take their coproduct cospan $A + C \to M + N \leftarrow B + D$. And for decorations, we use the map $\psi_{M,N} : \mathsf{Circ}(M) \times \mathsf{Circ}(N) \to \mathsf{Circ}(M + N)$. So, for example, suppose we want to take the monoidal product of the open circuits



and



The result is given by stacking them. In other words, their monoidal product is:



(6.87)

Easy, right?

We leave you to do two compositions of your own.

Exercise 6.88. Write x for the open circuit in (6.87). Also define cospans $\eta\colon 0 \to 2$ and $\eta\colon 2 \to 0$ as follows:

where each of these are decorated by the empty circuit $(\underline{1}, \varnothing, !, !, !) \in \mathsf{Circ}(\underline{1})$.[6]

Compute the composite $\eta \fatsemi x \fatsemi \epsilon$ in $\mathbf{Cospan}_{\mathsf{Circ}}$. This is a morphism $\underline{0} \to \underline{0}$; we call such things *closed circuits*. ◊

6.5 Operads and their algebras

In Theorem 6.77 we described how decorating cospans builds a hypergraph category from a symmetric monoidal functor. We then explored how that works in the case that the decoration functor is somehow "all circuit graphs on a set of nodes".

In this book, we have devoted a great deal of attention to different sorts of compositional theories, from monoidal preorders to compact closed categories to hypergraph categories. Yet for an application you someday have in mind, it may be the case that none of these theories suffice. You need a different structure, customized to a particular situation. For example in [VSL15] the authors wanted to compose continuous dynamical systems with control-theoretic properties and realized that in order for feedback to make sense, the wiring diagrams could not involve what they called 'passing wires'.

So to close our discussion of compositional structures, we want to quickly sketch something we can use as a sort of meta-compositional structure, known as an operad. We saw in Section 6.4.3 that we can build electric circuits from a symmetric monoidal functor $\mathbf{FinSet} \to \mathbf{Set}$. Similarly we'll see that we can build examples of new algebraic structures from operad functors $\mathcal{O} \to \mathbf{Set}$.

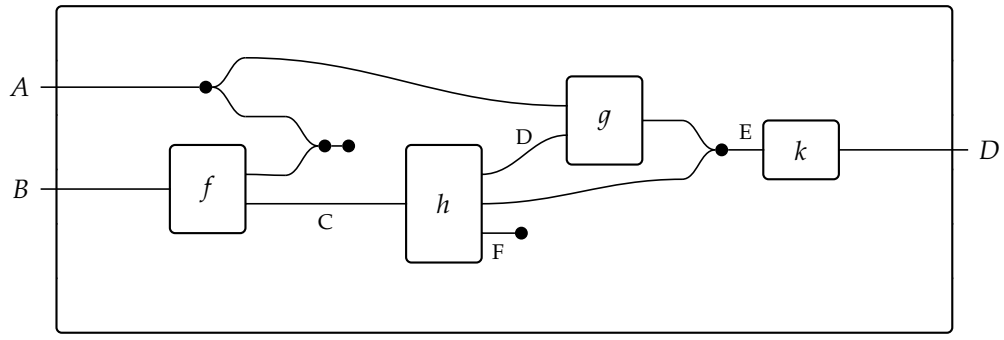
6.5.1 Operads design wiring diagrams

Understanding that circuits are morphisms in a hypergraph category is useful: it means we can bring the machinery of category theory to bear on understanding electrical

circuits. For example, we can build functors that express the compositionality of circuit semantics, i.e. how to derive the functionality of the whole from the functionality and interaction pattern of the parts. Or we can use the category-theoretic foundation to relate circuits to other sorts of network systems, such as signal flow graphs. Finally, the basic coherence theorems for monoidal categories and compact closed categories tell us that wiring diagrams give sound and complete reasoning in these settings.

However, one perhaps unsatisfying result is that the hypergraph category introduces artifacts like the domain and codomain of a circuit, which are not inherent to the structure of circuits or their composition. Circuits just have a single boundary interface, not 'domains' and 'codomains'. This is not to say the above model is not useful: in many applications, a vector space does not have a preferred basis, but it is often useful to pick one so that we may use matrices (or signal flow graphs!). But it would be worthwhile to have a category-theoretic model that more directly represents the compositional structure of circuits. In general, we want the category-theoretic model to fit our desired application like a glove. Let us quickly sketch how this can be done.

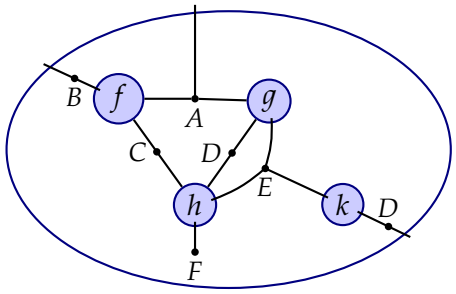
Let's return to wiring diagrams for a second. We saw that wiring diagrams for hypergraph categories basically look like this:



(6.89)

Note that if you had a box with A and B on the left and D on the right, you could plug the above diagram right inside it, and get a new open circuit. This is the basic move of operads.

But before we explain this, let's get where we said we wanted to go: to a model where there aren't ports on the left and ports on the right, there are just ports. We want a more succinct model of composition for circuit diagrams; something that looks more like this:



(6.90)

Do you see how diagrams Eq. (6.89) and Eq. (6.90) are actually exactly the same in terms of interconnection pattern? The only difference is that the latter does not have left/right distinction: we have lost exactly what we wanted to lose.

The cost is that the 'boxes' f, g, h, k in Eq. (6.90) no longer have a left/right distinction; they're just circles now. That wouldn't be bad except that it means they can no longer represent morphisms in a category—like they used to above, in Eq. (6.89)—because morphisms in a category by definition have a domain and codomain. Our new circles have no such distinction. So now we need a whole new way to think about 'boxes' categorically: if they're no longer morphisms in a category, what are they? The answer is found in the theory of operads.

In understanding operads, we will find we need to navigate one of the level shifts that we first discussed in Section 1.4.5. Notice that for decorated cospans, we define a hypergraph *category* using a symmetric monoidal *functor*. This is reminiscent of our brief discussion of algebraic theories in Section 5.4.2, where we defined something called the theory of monoids as a prop $\mathcal{M}$, and define monoids using functors $\mathcal{M} \to \mathbf{Set}$; see Remark 5.74. In the same way, we can view the category $\mathbf{Cospan}_{\mathbf{FinSet}}$ as some sort of 'theory of hypergraph categories', and so define hypergraph categories as functors $\mathbf{Cospan}_{\mathbf{FinSet}} \to \mathbf{Set}$.

So that's the idea. An operad $\mathcal{O}$ will define a theory or grammar of composition, and operad functors $\mathcal{O} \to \mathbf{Set}$, known as $\mathcal{O}$-*algebras*, will describe particular applications that obey that grammar.

Rough Definition 6.91. To specify an *operad* $\mathcal{O}$,

(i) one specifies a collection T, whose elements are called *types*;

(ii) for each tuple $(t_1, \ldots, t_n, t)$ of types, one specifies a set $\mathcal{O}(t_1, \ldots, t_n; t)$, whose elements are called *operations of arity* $(t_1, \ldots, t_n; t)$;

(iii) for each pair of tuples $(s_1, \ldots, s_m, t_i)$ and $(t_1, \ldots, t_n, t)$, one specifies a function

$$\circ_i \colon \mathcal{O}(s_1, \ldots, s_m; t_i) \times \mathcal{O}(t_1, \ldots, t_n; t) \to \mathcal{O}(t_1, \ldots, t_{i-1}, s_1, \ldots, s_m, t_{i+1}, \ldots, t_n; t);$$

called *substitution*; and

(iv) for each type t, one specifies an operation $\mathrm{id}_t \in O(t; t)$ called the *identity operation*.

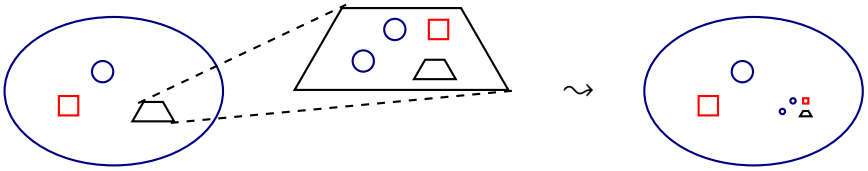
These must obey generalized identity and associativity laws.[7]

Let's ignore types for a moment and think about what this structure models. The intuition is that an operad consists of, for each n, a set of operations of arity n—that is, all the operations that accept n arguments. If we take an operation f of arity m, and plug the output into the ith argument of an operation g of arity n, we should get an operation of arity $m + n - 1$: we have m arguments to fill in m, and the remaining $n - 1$

[6] As usual ! denotes the unique function, in this case from the empty set to the relevant codomain.

[7] Often what we call types are called objects or colors, what we call operations are called morphisms, what we call substitution is called composition, and what we call operads are called multicategories. A formal definition can be found in [Lei04].

arguments to fill in g. Which operation of arity $m+n-1$ do we get? This is described by the substitution function $\circ_i$, which says we obtain the operation $f \circ_i g \in \mathcal{O}(m+n-1)$. The coherence conditions say that these functions $\circ_i$ capture the following intuitive picture:



The types then allow us to specify the, well, types of the arguments—inputs—that each function takes. So making tea is a 2-ary operation, an operation with arity 2, because it takes in two things. To make tea you need some warm water, and you need some tea leaves.

Example 6.92. Context-free grammars are to operads as graphs are to categories. Let's sketch what this means. First, a context-free grammar is a way of describing a particular set of 'syntactic categories' that can be formed from a set of symbols. For example, in English we have syntactic categories like nouns, determiners, adjectives, verbs, noun phrases, prepositional phrases, sentences, etc. The symbols are words, e.g. cat, dog, the, chases.

To define a context-free grammar on some alphabet, one specifies some *production rules*, which say how to form an entity in some syntactic category from a bunch of entities in other syntactic categories. For example, we can form a noun phrase from a determiner (the), an adjective (happy), and a noun (boy). Context free grammars are important in both linguistics and computer science. In the former, they're a basic way to talk about the structure of sentences in natural languages. In the latter, they're crucial when designing parsers for programming languages.

So just like graphs present free categories, context-free grammars present free operads. This idea was first noticed in [HMP98].

6.5.2 Operads from symmetric monoidal categories

We will see in Definition 6.97 that a large class of operads come from symmetric monoidal categories. Before we explain this, we give a couple of examples. Perhaps the most important operad is that of **Set**.

Example 6.93. The operad **Set** of sets has

(i) Sets X as types.

(ii) Functions $X_1 \times \cdots \times X_n \to Y$ as operations of arity $(X_1, \ldots, X_n; Y)$.

(iii) Substitution defined by

$$(g \circ_i f)(x_1, \ldots, x_{i-1}, w_1, \ldots, w_m, x_{i+1}, \ldots, x_n) \\ = g\big(x_1, \ldots, x_{i-1}, f(w_1, \ldots, w_m), x_{i+1}, \ldots, x_n\big)$$

where $f \in \mathbf{Set}(W_1, \ldots, W_m; X_i)$, $g \in \mathbf{Set}(X_1, \ldots, X_n; Y)$, and hence $g \circ_i f$ is a function

$$(g \circ_i f)\colon X_1 \times \cdots \times X_{i-1} \times W_1 \times \cdots \times W_m \times X_{i+1} \times \cdots \times X_n \longrightarrow Y$$

(iv) Identities $\mathrm{id}_X \in \mathbf{Set}(X; X)$ are given by the identity function $\mathrm{id}_X\colon X \to X$.

Next we give an example that reminds us what all this operad stuff was for: wiring diagrams.

Example 6.94. The operad **Cospan** of finite-set cospans has

(i) Natural numbers $a \in \mathbb{N}$ as types.

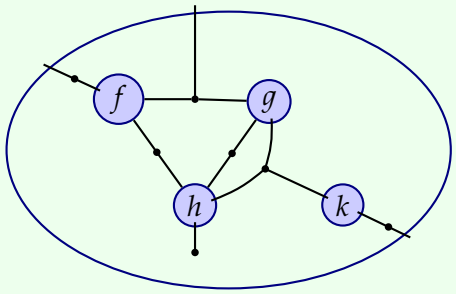
(ii) Cospans $\underline{a_1} + \cdots + \underline{a_n} \to \underline{p} \leftarrow \underline{b}$ of finite sets as operations of arity $(a_1, \ldots, a_n; b)$.

(iii) Substitution defined by pushout.

(iv) Identities $\mathrm{id}_a \in \mathbf{Set}(a; a)$ just given by the identity cospan $\underline{a} \xrightarrow{\mathrm{id}_{\underline{a}}} \underline{a} \xleftarrow{\mathrm{id}_{\underline{a}}} \underline{a}$.

This is the operadic analogue of the monoidal category $(\mathbf{Cospan}_{\mathbf{FinSet}}, 0, +)$.

We can depict operations in this operad using diagrams like we drew above. For example, here's a picture of an operation:



(6.95)

This is an operation of arity $(\underline{3}, \underline{3}, \underline{4}, \underline{2};\ \underline{3})$. Why? The circles marked f and g have 3 ports, h has 4 ports, k has 2 ports, and the outer circle has 3 ports: 3, 3, 4, 2; 3.

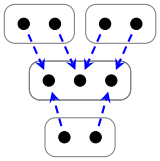
So how exactly is Eq. (6.95) a morphism in this operad? Well a morphism of this arity is, by (ii), a cospan $\underline{3} + \underline{3} + \underline{4} + \underline{2} \xrightarrow{a} \underline{p} \xleftarrow{b} \underline{3}$. In the diagram above, the apex $\underline{p}$ is the set $\underline{7}$, because there are 7 nodes $\bullet$ in the diagram. The function a sends each port on one of the small circles to the node it connects to, and the function b sends each port of the outer circle to the node it connects to.

We are able to depict each operation in the operad **Cospan** as a wiring diagram. It is often helpful to think of operads as describing a wiring diagram grammar. The

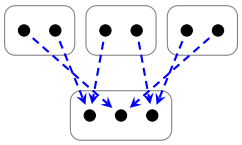
substitution operation of the operad signifies inserting one wiring diagram into a circle or box in another wiring diagram.

Exercise 6.96.

1. Consider the following cospan $f \in \mathbf{Cospan}(2,2;2)$:



   Draw it as a wiring diagram with two inner circles, each with two ports, and one outer circle with two ports.
2. Draw the wiring diagram corresponding to the following cospan $g \in \mathbf{Cospan}(2,2,2;0)$:



$\varnothing$

3. Compute the cospan $g \circ_1 f$. What is its arity?
4. Draw the cospan $g \circ_1 f$. Do you see it as substitution? ◊

We can turn any symmetric monoidal category into an operad in a way that generalizes the above two examples.

Definition 6.97. For any symmetric monoidal category $(\mathcal{C}, I, \otimes)$, there is an operad $\mathcal{O}_\mathcal{C}$, called the *operad underlying* $\mathcal{C}$, defined as having:

(i) $\mathrm{Ob}(\mathcal{C})$ as types.
(ii) morphisms $C_1 \otimes \cdots \otimes C_n \to D$ in $\mathcal{C}$ as the operations of arity $(C_1, \ldots, C_n; D)$.
(iii) substitution is defined by

$$(f \circ_i g) \coloneqq f \circ (\mathrm{id}, \ldots, \mathrm{id}, g, \mathrm{id}, \ldots, \mathrm{id})$$

(iv) identities $\mathrm{id}_a \in \mathcal{O}_\mathcal{C}(a;a)$ defined by id_a.

We can also turn any monoidal functor into what's called an operad functor.

6.5.3 The operad for hypergraph props

An operad functor takes the types of one operad to the types of another, and then the operations of the first to the operations of the second in a way that respects this.

Rough Definition 6.98. Suppose given two operads $\mathcal{O}$ and $\mathcal{P}$ with type collections T and U respectively. To specify an operad functor $F\colon \mathcal{O} \to \mathcal{P}$,

(i) one specifies a function $f\colon T \to U$.

(ii) For all arities $(t_1, \ldots, t_n; t)$ in $\mathcal{O}$, one specifies a function

$$F\colon \mathcal{O}(t_1, \ldots, t_n; t) \to \mathcal{P}(f(t_1), \ldots, f(t_n); f(t))$$

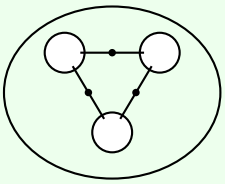
such that composition and identities are preserved.

Just as set-valued functors $\mathcal{C} \to \mathbf{Set}$ from any category $\mathcal{C}$ are of particular interest—we saw them as database instances in Chapter 3—so to are **Set**-valued functors $\mathcal{O} \to \mathbf{Set}$ from any operad $\mathcal{O}$.

Definition 6.99. An *algebra* for an operad $\mathcal{O}$ is an operad functor $F\colon \mathcal{O} \to \mathbf{Set}$.

We can think of functors $\mathcal{O} \to \mathbf{Set}$ as defining a set of possible ways to fill the boxes in a wiring diagram. Indeed, each box in a wiring diagram represents a type t of the given operad $\mathcal{O}$ and an algebra $F\colon \mathcal{O} \to \mathbf{Set}$ will take a type t and return a set $F(t)$ of fillers for box t. Moreover, given an operation (i.e., a wiring diagram) $f \in \mathcal{O}(t_1, \ldots, t_n; t)$, we get a function $F(f)$ that takes an element of each set $F(t_i)$, and returns an element of $F(t)$. For example, it takes n circuits with interface $t_1, \ldots, t_n$ respectively, and returns a circuit with boundary t.

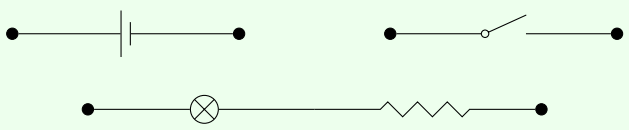
Example 6.100. For electric circuits, the types are again finite sets, $T = \mathrm{Ob}(\mathbf{FinSet})$, where each finite set $t \in T$ corresponds to a cell with t ports. Just as before, we have a set $\mathsf{Circ}(t)$ of fillers, namely the set of electric circuits with that t-marked terminals. As an operad algebra, $\mathsf{Circ}\colon \mathbf{Cospan} \to \mathbf{Set}$ transforms wiring diagrams like this one

$\varphi :=$ 

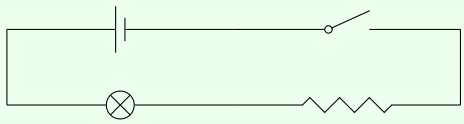
into formulas that build a new circuit from a bunch of existing ones. In the above-drawn case, we would get a morphism $\mathsf{Circ}(\varphi) \in \mathbf{Set}(\mathsf{Circ}(2), \mathsf{Circ}(2), \mathsf{Circ}(2); \mathsf{Circ}(0))$, i.e. a function

$$\mathsf{Circ}(\varphi)\colon \mathsf{Circ}(2) \times \mathsf{Circ}(2) \times \mathsf{Circ}(2) \to \mathsf{Circ}(0).$$

We could apply this function to the three elements of $\mathsf{Circ}(2)$ shown here



and the result would be the closed circuit from the beginning of the chapter:



This is reminiscent of the story for decorated cospans: gluing fillers together to form hypergraph categories. An advantage of the decorated cospan construction is that one obtains an explicit category (where morphisms have domains and codomains and can hence be composed associatively), equipped with Frobenius structures that allow us to get around the strictures of domains and codomains. The operad perspective has other advantages. First, whereas decorated cospans can produce only some hypergraph categories, **Cospan**-algebras can produce any hypergraph category.

Proposition 6.101. There is an equivalence between **Cospan**-algebras and hypergraph props.

Another advantage of using operads is that one can vary the operad itself, from **Cospan** to something similar (like the operad of 'cobordisms'), and get slightly different compositionality rules.

In fact, operads—with the additional complexity in their definition—can be customized even more than all compositional structures defined so far. For example, we can define operads of wiring diagrams where the wiring diagrams must obey precise conditions far more specific than the constraints of a category, such as requiring that the diagram itself has no wires that pass straight through it. In fact, operads are strong enough to define themselves: roughly speaking, there is an operad for operads: the category of operads is equivalent to the category of algebras for a certain operad [Lei04, Example 2.2.23]. While operads can, of course, be generalized again, they conclude our march through an informal hierarchy of compositional structures, from preorders to categories to monoidal categories to operads.

6.6 Summary and further reading

This chapter began with a detailed exposition of colimits in the category of sets; as we saw, these colimits describe ways of joining or interconnecting sets. Our second way of talking about interconnection was the use of Frobenius monoids and hypergraph categories; we saw these two themes come together in the idea of a decorated cospans. The decorated cospan construction uses a certain type of structured functor to construct a certain type of structured category. More generally, we might be interested in other types of structured category, or other compositional structure. To address this, we briefly saw how these ideas fit into the theory of operads.

Colimits are a fundamental concept in category theory. For more on colimits, one might refer to any of the introductory category theory textbooks we mentioned in Section 3.6.

Special commutative Frobenius monoids and hypergraph categories were first defined, under the names 'separable commutative Frobenius algebra' and 'well-supported compact closed category', by Carboni and Walters [CW87; Car91]. The use of decorated cospans to construct them is detailed in [Fon15; Fon18; Fon16]. The application to networks of passive linear systems, such as certain electrical circuits, is discussed in [BF15], while further applications, such as to Markov processes and chemistry can be found in [BFP16; BP17]. For another interesting application of hypergraph categories, we recommend the pixel array method for approximating solutions to nonlinear equations [Spi+16]. The story of this chapter is fleshed out in a couple of recent, more technical papers [FS18b; FS18a].

Operads were introduced by May to describe compositional structures arising in algebraic topology [May72]; Leinster has written a great book on the subject [Lei04]. More recently, with collaborators author-David has discussed using operads in applied mathematics, to model composition of structures in logic, databases, and dynamical systems [RS13; Spi13; VSL15].

18.S097 Applied Category Theory

January IAP 2019