
Preface

Category theory is becoming a central hub for all of pure mathematics. It is unmatched in its ability to organize and layer abstractions, to find commonalities between structures of all sorts, and to facilitate communication between different mathematical communities.

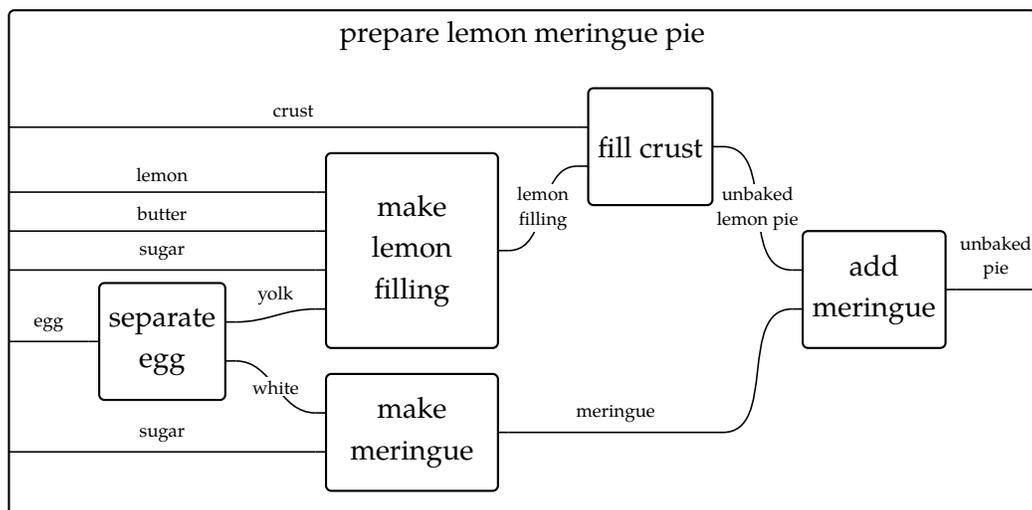
But it has also been branching out into science, informatics, and industry. We believe that it has the potential to be a major cohesive force in the world, building rigorous bridges between disparate worlds, both theoretical and practical. The motto at MIT is *mens et manus*, Latin for mind and hand. We believe that category theory—and pure math in general—has stayed in the realm of mind for too long; it is ripe to be brought to hand.

Purpose and audience

The purpose of this book is to offer a self-contained tour of applied category theory. It is an invitation to discover advanced topics in category theory through concrete real-world examples. Rather than try to give a comprehensive treatment of these topics—which include adjoint functors, enriched categories, proarrow equipments, toposes, and much more—we merely provide a taste. We want to give readers some insight into how it feels to work with these structures as well as some ideas about how they might show up in practice.

The audience for this book is quite diverse: anyone who finds the above description intriguing. This could include a motivated high school student who hasn't seen calculus yet but has loved reading a weird book on mathematical logic they found at the library. Or a machine learning researcher who wants to understand what vector spaces, design theory, and dynamical systems could possibly have in common. Or a pure mathematician who wants to imagine what sorts of applications their work might have. Or a recently-retired programmer who's always had an eerie feeling that category theory is what they've been looking for to tie it all together, but who's found the usual books on the subject impenetrable.

For example, we find it something of a travesty that in 2018 there seems to be no introductory material available on monoidal categories. Even beautiful modern introductions to category theory, e.g. by Riehl or Leinster, do not include anything on this rather central topic. The basic idea is certainly not too abstract; modern human intuition seems to include a pre-theoretical understanding of monoidal categories that is just waiting to be formalized. Is there anyone who wouldn't correctly understand the basic idea being communicated in the following diagram?



Many applied category theory topics seem to take monoidal categories as their jumping-off point. So one aim of this book is to provide a reference—even if unconventional—for this important topic.

We hope this book inspires both new visions and new questions. We intend it to be self-contained in the sense that it is approachable with minimal prerequisites, but not in the sense that the complete story is told here. On the contrary, we hope that readers use this as an invitation to further reading, to orient themselves in what is becoming a large literature, and to discover new applications for themselves.

This book is, unashamedly, our take on the subject. While the abstract structures we explore are important to any category theorist, the specific topics have simply been chosen to our personal taste. Our examples are ones that we find simple but powerful, concrete but representative, entertaining but in a way that feels important and expansive at the same time. We hope our readers will enjoy themselves and learn a lot in the process.

How to read this book

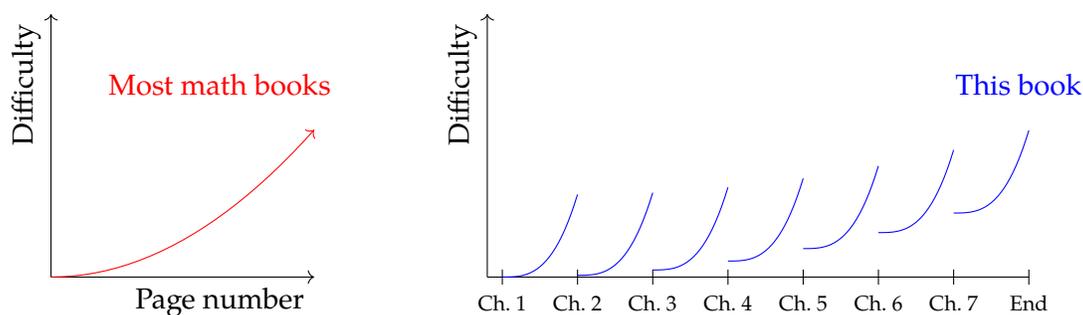
The basic idea of category theory—which threads through every chapter—is that if one pays careful attention to structures and coherence, the resulting systems will be extremely reliable and interoperable. For example, a category involves several structures: a collection of objects, a collection of morphisms relating objects, and a

formula for combining any chain of morphisms into a morphism. But these structures need to *cohere* or work together in a simple commonsense way: a chain of chains is a chain, so combining a chain of chains should be the same as combining the chain. That's it!

We will see structures and coherence come up in pretty much every definition we give: “here are some things and here are how they fit together.” We ask the reader to be on the lookout for structures and coherence as they read the book, and to realize that as we layer abstraction on abstraction, it is the coherence that makes everything function like a well-oiled machine.

Each chapter in this book is motivated by a real-world topic, such as electrical circuits, control theory, cascade failures, information integration, and hybrid systems. These motivations lead us into and through various sorts of category-theoretic concepts. We generally have one motivating idea and one category-theoretic purpose per chapter, and this forms the title of the chapter, e.g. Chapter 4 is “Collaborative design: profunctors, categorification, and monoidal categories.”

In many math books, the difficulty is roughly a monotonically-increasing function of the page number. In this book, this occurs in each chapter, but not so much in the book as a whole. The chapters start out fairly easy and progress in difficulty.



The upshot is that if you find the end of a chapter very difficult, hope is certainly not lost: you can start on the next one and make good progress. This format lends itself to giving you a first taste now, but also leaving open the opportunity for you to come back at a later date and get more deeply into it. But by all means, if you have the gumption to work through each chapter to its end, we very much encourage that!

We include many exercises throughout the text. Usually these exercises are fairly straightforward; the only thing they demand is that the reader's mind changes state from passive to active, rereads the previous paragraphs with intent, and puts the pieces together. A reader becomes a student when they work the exercises; until then they are more of a tourist, riding on a bus and listening off and on to the tour guide. Hey, there's nothing wrong with that, but we do encourage you to get off the bus and make contact with the natives as often as you can.

Acknowledgments

Thanks to Peter Gates, Nelson Niu, Samantha Seaman, Adam Theriault-Shay, Andrew Turner for many helpful comments and conversations. We also thank Ramón Jaramillo for designing and making available the circuit diagram for an amplifier at the beginning of Chapter 6.

Contents

1	Generative effects: Posets and adjunctions	1
1.1	More than the sum of their parts	1
1.1.1	A first look at generative effects	2
1.1.2	Ordering systems	4
1.2	Posets	6
1.2.1	Review of sets, relations, and functions	6
1.2.2	Posets	10
1.3	Monotone maps	13
1.4	Meets and joins	17
1.4.1	Definition and basic examples	17
1.4.2	Back to observations and generative effects	18
1.5	Galois connections	19
1.5.1	Definition and examples of Galois connections	19
1.5.2	Back to partitions	20
1.5.3	Basic theory of Galois connections	22
1.5.4	Closure operators	25
1.5.5	Level shifting	27
1.6	Summary and further reading	28
2	Resources: monoidal posets and enrichment	29
2.1	Getting from a to b	29
2.2	Symmetric monoidal posets	31
2.2.1	Definition and first examples	31
2.2.2	Introducing wiring diagrams	33
2.2.3	Applied examples	37
2.2.4	Abstract examples	41
2.2.5	Monoidal monotone maps	44
2.3	Enrichment	45
2.3.1	\mathcal{V} -categories	45
2.3.2	Posets as Bool -categories	46

2.3.3	Lawvere metric spaces	47
2.3.4	\mathcal{V} -variations on posets and metric spaces	51
2.4	Constructions on enriched categories	51
2.4.1	Changing the base of enrichment	52
2.4.2	Enriched functors	52
2.4.3	Product \mathcal{V} -categories	53
2.5	Computing presented \mathcal{V} -categories with matrix mult.	55
2.5.1	Monoidal closed posets	55
2.5.2	Commutative quantales	57
2.5.3	Matrix multiplication in a quantale	59
2.6	Summary and further reading	61
3	Databases: Categories, functors, and (co)limits	63
3.1	What is a database?	63
3.2	Categories	67
3.2.1	Free categories	67
3.2.2	Presenting categories via path equations	69
3.2.3	Posets and free categories: two ends of a spectrum	70
3.2.4	Important categories in mathematics	72
3.2.5	Isomorphisms in a category	73
3.3	Functors, natural transformations, and databases	74
3.3.1	Sets and functions as databases	74
3.3.2	Functors	75
3.3.3	Databases as Set -valued functors	77
3.3.4	Natural transformations	79
3.3.5	The category of instances on a schema	81
3.4	Adjunctions and data migration	83
3.4.1	Pulling back data along a functor	84
3.4.2	Adjunctions	86
3.4.3	Left and right pushforward functors, Σ and Π	87
3.4.4	Single set summaries of databases	89
3.5	Bonus: An introduction to limits and colimits	90
3.5.1	Terminal objects and products	90
3.5.2	Limits	93
3.5.3	Finite limits in Set	94
3.5.4	A brief note on colimits	95
3.6	Summary and further reading	96
4	Co-design: profunctors and monoidal categories	99
4.1	Can we build it?	99
4.2	Enriched profunctors	101
4.2.1	Feasibility relationships as Bool -profunctors	101

4.2.2	\mathcal{V} -profunctors	103
4.2.3	Back to co-design diagrams	106
4.3	Categories of profunctors	107
4.3.1	Composing profunctors	107
4.3.2	The categories \mathcal{V} - Prof and Feas	109
4.3.3	Fun profunctor facts: companions, conjoints, collages	111
4.4	Categorification	113
4.4.1	The basic idea of categorification	114
4.4.2	A reflection on wiring diagrams	115
4.4.3	Monoidal categories	117
4.4.4	Categories enriched in a symmetric monoidal category	119
4.5	Profunctors form a compact closed category	120
4.5.1	Compact closed categories	121
4.5.2	Feas as a compact closed category	124
4.6	Summary and further reading	125
5	Signal flow graphs: Props, presentations, & proofs	127
5.1	Comparing systems as interacting signal processors	127
5.2	Props and presentations	129
5.2.1	Props: definition and first examples	129
5.2.2	The prop of port graphs	130
5.2.3	Free constructions and universal properties	132
5.2.4	Free props	134
5.2.5	Props via presentations	136
5.3	Simplified signal flow graphs	137
5.3.1	Rigs	137
5.3.2	The iconography of signal flow graphs	138
5.3.3	The prop of matrices over a rig	141
5.3.4	Turning signal flow graphs into matrices	142
5.3.5	The idea of functorial semantics	145
5.4	Graphical linear algebra	145
5.4.1	A presentation of $\mathbf{Mat}(R)$	146
5.4.2	Aside: monoid objects in a monoidal category	148
5.4.3	Signal flow graphs: feedback and more	150
5.5	Summary and further reading	154
6	Circuits: hypergraph categories and operads	157
6.1	The ubiquity of network languages	157
6.2	Colimits and connection	160
6.2.1	Initial objects	160
6.2.2	Coproducts	161
6.2.3	Pushouts	163

6.2.4	Finite colimits	165
6.2.5	Cospans	167
6.3	Hypergraph categories	170
6.3.1	Special commutative Frobenius monoids	170
6.3.2	Wiring diagrams for hypergraph categories	173
6.3.3	Definition of hypergraph category	173
6.4	Decorated cospans	175
6.4.1	Symmetric monoidal functors	176
6.4.2	Decorated cospans	176
6.4.3	Electric circuits	179
6.5	Operads and their algebras	181
6.5.1	Operads design wiring diagrams	181
6.5.2	Operads from symmetric monoidal categories	184
6.5.3	The operad for hypergraph props	185
6.6	Summary and further reading	186
7	Logic of behavior: Sheaves, toposes, languages	189
7.1	How can we prove our machine is safe?	189
7.2	The category Set as an exemplar topos	192
7.2.1	Set -like properties enjoyed by any topos	193
7.2.2	The subobject classifier	195
7.2.3	Logic in the topos Set	197
7.3	Sheaves	199
7.3.1	Presheaves	199
7.3.2	Topological spaces	201
7.3.3	Sheaves on topological spaces	203
7.4	Toposes	208
7.4.1	The subobject classifier Ω in a sheaf topos	209
7.4.2	Logic in a sheaf topos	210
7.4.3	Predicates	211
7.4.4	Quantification	212
7.4.5	Modalities	214
7.4.6	Type theories and semantics	215
7.5	A topos of behavior types	215
7.5.1	The interval domain	215
7.5.2	Sheaves on \mathbb{IR}	216
7.5.3	Safety proofs in temporal logic	218
7.6	Summary and further reading	219
	Bibliography	221
	Index	229

Chapter 1

Generative effects: Posets and Galois connections

In this book, we explore a wide variety of situations—in the world of science, engineering, and commerce—where we see something we might call *compositionality*. These are cases in which systems or relationships can be combined to form new systems or relationships. In each case we find category-theoretic constructs—developed for their use in pure math—which beautifully describe these compositional aspects.

This chapter, being the first, must serve this goal in two capacities. First, it must provide motivating examples of this sort of compositionality, as well as the relevant categorical formulations. Second, it must provide the mathematical foundation for the rest of the book. Since we are starting with minimal assumptions about the reader’s background, we must begin slowly and build up throughout the book. As a result, examples in the early chapters are necessarily simplified. However, we hope the reader will already begin to see the sort of structural approach to modeling that category theory brings to the fore.

1.1 More than the sum of their parts

We motivate this first chapter by noticing that while many real-world structures are compositional, the results of observing them are often not. The reason is that observation is inherently “lossy”: in order to extract information from something, one must drop the details. For example, one stores a real number by rounding it to some precision. But if the details are actually relevant in a given system operation, then the observed result of that operation will not be as expected. This is clear in the case of roundoff error, but it also shows up in non-numerical domains: observing a complex system is rarely enough to predict its behavior because the observation is lossy.

A central theme in category theory is the study of structures and structure-preserving maps. A map $X \xrightarrow{m} Y$ is a kind of observation of object named X in another object named Y . Asking which structures in X one wants to preserve under m becomes the

question “what category are you working in?”. As an example, there are many functions from \mathbb{R} to \mathbb{R} , but only some of them are order-preserving, only some of them are metric-preserving, only some of them are addition-preserving, etc.

Exercise 1.1. A function $f: \mathbb{R} \rightarrow \mathbb{R}$ is

1. *order-preserving* if $x \leq y$ implies $f(x) \leq f(y)$, for all $x, y \in \mathbb{R}$;
2. *metric-preserving* if $|x - y| = |f(x) - f(y)|$;
3. *addition-preserving* if $f(x + y) = f(x) + f(y)$.

In each of the three cases above, find an f that is *foo*-preserving and an example of an f that is not *foo*-preserving. \diamond

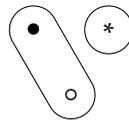
In category theory we want to keep control over which aspects of our systems are being preserved under our observation. As we said above, the less structure is preserved by our observation of a system, the more “surprises” occur when we observe its operations. One might call these surprised *generative effects*.

In using category theory to explore generative effects, we follow the basic ideas from work by Adam [Ada17]. He goes much more deeply into the issue than we can here; see Section 1.6. But as mentioned above, we must also use this chapter to give an order-theoretic warm-up for the full-fledged category theory to come.

1.1.1 A first look at generative effects

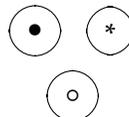
To explore the notion of a generative effect we need a sort of system, a sort of observation, and a system-level operation that is not preserved by the observation. Let’s start with a simple example.

Consider three points; we’ll call them \bullet , \circ and $*$. In this example, a system will simply be a way of connecting these points together. We might think of our points as sites on a power grid, with a system describing connection by power lines, or as people susceptible to some disease, with a system describing possible contagion. As an abstract example of a system, there is a system where \bullet and \circ are connected, but neither are connected to $*$. We shall draw this like so:

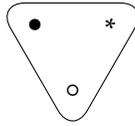


Connections are symmetric, so if a is connected to b , then b is connected to a . Connections are also transitive, meaning that if a is connected to b , and b is connected to c , then a is connected to c —that is, all a , b , and c are connected. Friendship is not transitive—my friend’s friend is not necessarily my friend—but word-of-mouth is.

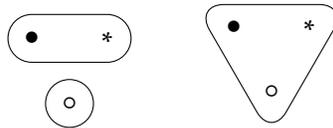
Let’s depict two more systems. Here is a system in which none of the points are connected:



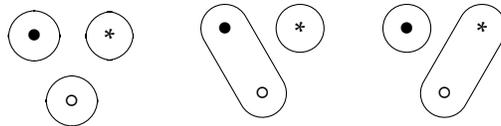
And here is the system in which all three points are connected:



Now that we have defined the sort of system we want to discuss, let us explain how we will observe these systems. Our observation Φ will be whether one point, say \bullet , is connected to another point, say $*$. An observation of the system will be an assignment of either true or false; we will assign true if \bullet is connected to $*$, and false otherwise. We thus assign the value true to the systems:

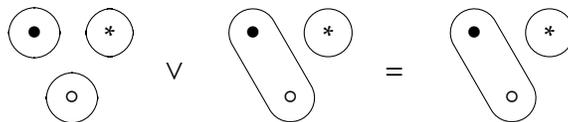


and we will assign the value false to the remaining systems:

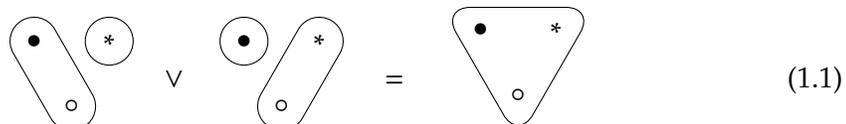


The last piece of setup is to give a sort of operation that can be done on systems. Let's call the operation "join". If the reader has been following the story arc, the expectation here is that our connectivity observation will not be compositional with respect to system joining; that is, there will be generative effects. Let's see what this means.

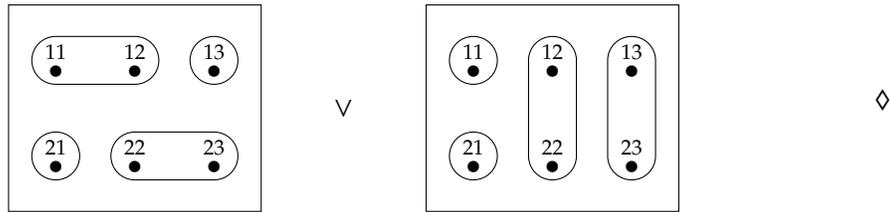
The join of two systems A and B is given simply by combining their connections. That is, we shall say the *join* of systems A and B , denote it $A \vee B$, has a connection between points x and y if there are some points z_1, \dots, z_n such that, in at least one of A or B , it is true that x is connected to z_1 , z_i is connected to z_{i+1} , and z_n is connected to y . In a three-point system, the above definition is overkill, but we want to say something that works for systems with any number of elements. The high-level way to say it is "take the transitive closure of the union of the connections in A and B ." In our three-element system, it means for example that



and



Exercise 1.2. What is the result of joining the following two systems?



We are now ready to see the generative effect. We don't want to build it up too much—this example has been made as simple as possible—but we will see that our observation fails to preserve the join operation. Let's denote our observation, measuring whether \bullet is connected to $*$, by Φ ; it returns a boolean result, either `true` or `false`.

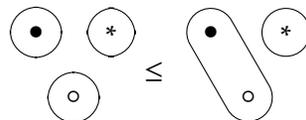
In the second example above, Eq. (1.1) we see that $\Phi(\mathcal{V}) = \Phi(\mathcal{W}) = \text{false}$: in both cases \bullet is not connected to $*$. On the other hand, when we join these two systems, we see that $\Phi(\mathcal{V} \vee \mathcal{W}) = \Phi(\mathcal{X}) = \text{true}$: in the joined system, \bullet is connected to $*$. There is no operation on the booleans `true`, `false` that will always follow suit with the joining of systems: our observation is inherently lossy.

While this was a simple example, it should be noted that whether the potential for such effects exist—i.e. determining whether an observation is operation-preserving—can be incredibly important information to know. For example, the two constituent systems could be the views of two local authorities on possibly contagion between an infected person \bullet and a vulnerable person $*$. They can either separately extract information from their raw data and then combine the results, or combine their raw data and extract information from it, and these give different answers.

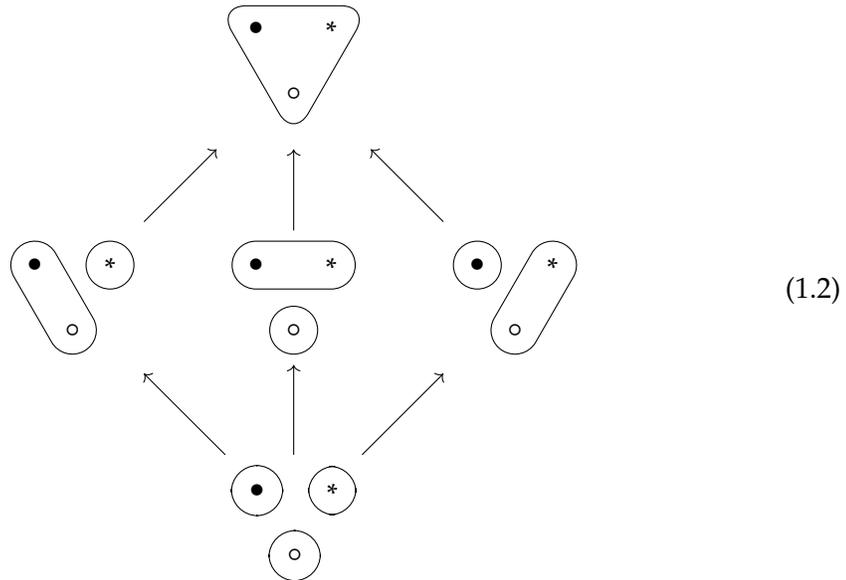
1.1.2 Ordering systems

Category theory is all about organizing and layering structures. In this section we will explain how the operation of joining systems can be derived from a more basic structure: order. We will see that while joining is not preserved by our connectivity observation Φ , order is.

To begin, we note that the systems themselves are ordered in a hierarchy. Given systems A and B , we say that $A \leq B$ if, whenever x is connected to y in A , then x is connected to y in B . For example,



This notion of \leq leads to the following diagram:



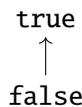
where we draw an arrow from system A to system B if $A \leq B$. Such diagrams are known as *Hasse diagrams*.

As we were saying above, the notion of join is derived from this order. Indeed for any two systems A and B in the Hasse diagram (1.2), the joined system $A \vee B$ is the smallest system that is bigger than both A and B . That is, $A \leq (A \vee B)$ and $B \leq (A \vee B)$, and for any C , if $A \leq C$ and $B \leq C$ then $(A \vee B) \leq C$. Let's walk through this with an exercise.

- Exercise 1.3.*
1. Write down all the partitions of a two element set $\{\bullet, *\}$, order them as above, and draw the Hasse diagram.
 2. Now do the same thing for a four element-set, say $\{1, 2, 3, 4\}$. There should be 15 partitions.
- Choose any two systems in your 15-element Hasse diagram, call them A and B .
3. What is $A \vee B$, using the definition given in the paragraph above Eq. (1.1)?
 4. Is it true that $A \leq (A \vee B)$ and $B \leq (A \vee B)$?
 5. What are all the systems C for which both $A \leq C$ and $B \leq C$.
 6. Is it true that in each case, $(A \vee B) \leq C$?

◇

The set $\mathbb{B} = \{\text{true}, \text{false}\}$ of booleans also has an order, $\text{false} \leq \text{true}$:



Thus $\text{false} \leq \text{false}$, $\text{false} \leq \text{true}$, and $\text{true} \leq \text{true}$, but $\text{true} \not\leq \text{false}$. In other words $A \leq B$ in the poset if “ A implies B ”, often denoted $A \Rightarrow B$.

For any A, B in $\mathbb{B}\{\text{true}, \text{false}\}$, we can again write $A \vee B$ to mean the least element that is greater than both A and B .

Exercise 1.4. In the order $\text{false} \leq \text{true}$ on $\mathbb{B} = \{\text{true}, \text{false}\}$, what is:

1. $\text{true} \vee \text{false}$?
2. $\text{false} \vee \text{true}$?
3. $\text{true} \vee \text{true}$?
4. $\text{false} \vee \text{false}$?

◇

Let's return to our systems with \bullet , \circ , and $*$, and our " \bullet is connected to $*$ " function, which we called Φ . It takes any such system and returns either `true` or `false`. Note that the map Φ preserves the \leq order: if $A \leq B$ and there is a connection between \bullet and $*$ in A , then there is such a connection in B too. The existence of the generative effect, however, is captured in the inequality

$$\Phi(A) \vee \Phi(B) \leq \Phi(A \vee B). \quad (1.3)$$

This can be a strict inequality: we showed two systems A and B with $\Phi(A) = \Phi(B) = \text{false}$, so $\Phi(A) \vee \Phi(B) = \text{false}$, but where $\Phi(A \vee B) = \text{true}$.

These ideas capture the most basic ideas in category theory. Most directly, we have seen that the map Φ preserves some structure but not others: order but not join. But in fact, we have seen here hints of more complex notions from category theory, without making them explicit; these include the notions of category, functor, colimit, and adjunction. In this chapter we will explore these ideas in the elementary setting of ordered sets.

1.2 Posets

Above we informally spoke of two different ordered sets: the order on system connectivity and the order on booleans $\text{false} \leq \text{true}$. Then we related these two ordered sets by means of an observation Φ . Before continuing, we need to make such ideas more precise. We begin in Section 1.2.1 with a review of sets and relations. In Section 1.2.2 we will give the definition of a poset—a preordered set—and a good number of examples.

1.2.1 Review of sets, relations, and functions

We will not give a definition of *set* here, but informally we will think of a set as a collection of things, known as elements. These things could be all the leaves on a tree, or the names of your favorite fruits, or simply some symbols a, b, c . For example, we write $A = \{h, 1\}$ to denote the set, called A , that contains exactly two elements, one called h and one called 1 . For an arbitrary set X , we write $x \in X$ if x is element of X ; so we have $h \in A$ and $1 \in A$, but $0 \notin A$.

Given sets X and Y , we say that X is a *subset* of Y , and write $X \subseteq Y$, if every element in X is also in Y . For example $\{h\} \subseteq A$. Note that the empty set $\emptyset := \{\}$ is a subset of every other set.

Given two sets X and Y , the product $X \times Y$ of X and Y is the set of pairs (x, y) , where $x \in X$ and $y \in Y$.

Exercise 1.5. Let $A := \{h, 1\}$ and $B := \{1, 2, 3\}$.

1. There are eight subsets of B ; write them out.
2. There are six elements in $A \times B$; write them out.

◇

A particularly important sort of subset is a subset of the product of a set with itself.

Definition 1.6. Let X and Y be sets. A *relation between X and Y* is a subset $R \subseteq X \times Y$. A *binary relation on X* is a relation between X and X , i.e. a subset $R \subseteq X \times X$.

It is convenient to use something called *infix notation* for binary relations $R \subseteq A \times A$. This means one picks a symbol, say \star , and writes $a \star b$ to mean $(a, b) \in R$.

Example 1.7. There is a binary relation on \mathbb{R} with infix notation \leq . Rather than writing $(5, 6) \in R$, we write $5 \leq 6$.

Other examples of infix notation for relations are $=, \approx, <, >$. In number theory, they are interested in whether one number divides without remainder into another; this relation is denoted with infix notation $|$, so $5|10$. ◇

Partitions and equivalence relations We can now define partitions more formally.

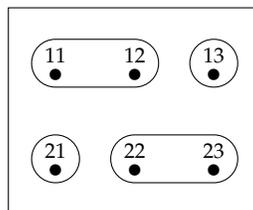
Definition 1.8. If A is a set, a *partition* of A consists of a set P and, for each $p \in P$, a nonempty subset $A_p \subseteq A$, such that

$$A = \bigcup_{p \in P} A_p \quad \text{and} \quad \text{if } p \neq q \text{ then } A_p \cap A_q = \emptyset \quad (1.4)$$

We may denote the partition by $\{A_p\}_{p \in P}$. We refer to P as the set of *part labels* and if $p \in P$ is a part label, we refer to A_p as the *p th part*. The condition (1.4) says that each element $a \in A$ is in exactly one part.

Let $a_1, a_2 \in A$ be elements. Given a partition, we define a relation with infix notation \sim , where $a_1 \sim a_2$ iff a and b are in the same part.¹

Exercise 1.9. Consider the partition shown below:



¹The notation “iff” means *if and only if*. So here we are saying that $a \sim b$ if a and b are both in the same part, and $a \not\sim b$ if a and b are in different parts.

Write down every pair (a, b) such that $a \sim b$. There should be 10. \diamond

We will see in Proposition 1.11 that there is a strong relationship between partitions and something called equivalence relations, which we define next.

Definition 1.10. Let A be a set. An *equivalence relation* on A is a binary relation, let's give it infix notation \sim , satisfying the following three properties:

1. $a \sim a$, for all $a \in A$,
2. $a \sim b$ iff $b \sim a$, for all $a, b \in A$, and
3. if $a \sim b$ and $b \sim c$ then $a \sim c$, for all $a, b, c \in A$.

These properties are called *reflexivity*, *symmetry*, and *transitivity*, respectively.

Proposition 1.11. Let A be a set. There is a one-to-one correspondence between the ways to partition A and the equivalence relations on A .

Proof. We first show that every partition gives rise to an equivalence relation, and then that every equivalence relation gives rise to a partition. Our two constructions will be mutually inverse, proving the proposition.

Suppose given a partition $\{A_p\}_{p \in P}$; we define a relation \sim and show it is an equivalence relation. Define $a \sim b$ to mean that a and b are in the same part: there is some $p \in P$ such that $a \in A_p$ and $b \in A_p$. It is obvious that a is in the same part as itself. Similarly, it is obvious that if a is in the same part as b then b is in the same part as a , and that if further b is in the same part as c then a is in the same part as c . Thus \sim is an equivalence relation.

Suppose given an equivalence relation \sim ; we will form a partition on A by saying what the parts are. Say that a subset $X \subseteq A$ is (\sim) -closed if, for every $x \in X$ and $x' \sim x$, we have $x' \in X$. Say that a subset $X \subseteq A$ is (\sim) -connected if it is nonempty and $x \sim y$ for every $x, y \in X$. Then the parts corresponding to \sim are exactly the (\sim) -closed, (\sim) -connected subsets. It is not hard to check that these indeed form a partition. \square

Exercise 1.12. Consider the proof of Proposition 1.11. Suppose that \sim is an equivalence relation, and let P be the set of (\sim) -closed and (\sim) -connected subsets $(A_p)_{p \in P}$.

1. Show that each A_p is nonempty.
2. Show that if $p \neq q$, i.e. if A_p and A_q are not exactly the same set, then $A_p \cap A_q = \emptyset$.
3. Show that $A = \bigcup_{p \in P} A_p$. \diamond

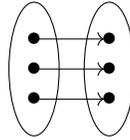
Functions The most frequently used sort of relation between sets is that of functions.

Definition 1.13. Let S and T be sets. A *function* from S to T is a subset $F \subseteq S \times T$ such that for all $s \in S$ there exists a unique $t \in T$ with $(s, t) \in F$.

The function F is often denoted $F: S \rightarrow T$. From now on, we write $F(s) = t$ to mean $(s, t) \in F$. For any $t \in T$, the *preimage* of t along F is the subset $\{s \in S \mid f(s) = t\}$.

The function is called *surjective* if it satisfies the converse of existence: for all $t \in T$ there exists $s \in S$ with $F(s) = t$. The function is called *injective* if it satisfies the converse of uniqueness: for all $t \in T$ and $s_1, s_2 \in S$, if $F(s_1) = t$ and $F(s_2) = t$ then $s_1 = s_2$.

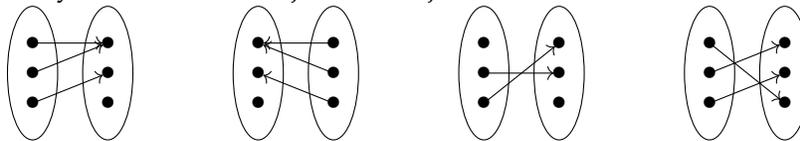
Example 1.14. An important but very simple sort of function is the *identity function* on a set X , denoted id_X . It is the function $\text{id}_X(x) = x$.



◆

Exercise 1.15. Answer the following questions for each of the relations below

1. Is it a function?
2. If not, why not? If so, is it injective, surjective, both, or neither?



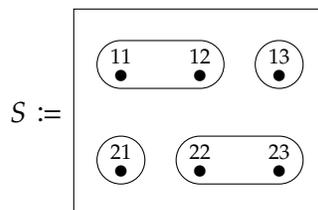
3. Find two sets A and B and a function $f: A \rightarrow B$ that is injective but not surjective.
4. Find two sets A and B and a function $f: B \rightarrow A$ that is surjective but not injective.

◆

Remark 1.16. A partition on a set A can also be understood in terms of surjective functions out of A . Given a surjective function $f: A \rightarrow P$, where P is any other set, consider the preimages $f^{-1}(p) \subseteq A$, one for each element $p \in P$. These partition A .

Note that if P' is another set isomorphic to P (same number of elements, possibly different names), say $i: P \xrightarrow{\cong} P'$ then of course the composite $A \xrightarrow{f} P \xrightarrow{i} P'$ will give the same partition. So we only care about P “up to isomorphism”.

Example 1.17. Here is an example of how to define a partition using a surjective function. Consider the partition of $S := \{11, 12, 13, 21, 22, 23\}$ shown below:



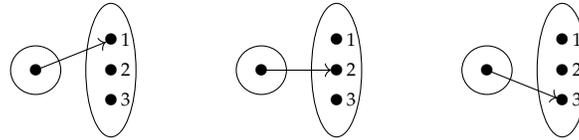
It has been partitioned into four parts, so let $P = \{a, b, c, d\}$ and let $f: S \rightarrow P$ be given by

$$f(11) = f(12) = a, \quad f(13) = b, \quad f(21) = c, \quad f(22) = d, \quad f(23) = d \quad \blacklozenge$$

Exercise 1.18. Write down the surjection corresponding to each of the five partitions in Eq. (1.2). ◆

Definition 1.19. If $F: X \rightarrow Y$ is a function and $G: Y \rightarrow Z$ is a function, their *composite* is the function $X \rightarrow Z$ defined to be $G(F(x))$ for any $x \in X$. It is often denoted $G \circ F$, but we prefer to denote it $F.G$. It takes any element $x \in X$, evaluates F to get an element $F(x) \in Y$ and then evaluates G to get an element $G(F(x))$.

Example 1.20. If X is any set and $x \in X$ is any element, we can think of x as a function $\{1\} \rightarrow X$, namely the function sending 1 to x . For example, the three functions $\{1\} \rightarrow \{1, 2, 3\}$ shown below correspond to the three elements of $\{1, 2, 3\}$:



Suppose given a function $F: X \rightarrow Y$ and an element of X , thought of as a function $x: \{1\} \rightarrow X$. Then evaluating F at x is given by the composite $x.F$. \blacklozenge

1.2.2 Posets

In Section 1.1, we several times used the symbol \leq to denote a sort of order. Here is a formal definition of what it means for a set to have an order.

Definition 1.21. A *preorder* on a set X is a binary relation X with infix notation \leq , such that

1. $x \leq x$; and
2. if $x \leq y$ and $y \leq z$, then $x \leq z$.

The first condition is called *reflexivity* and the second is called *transitivity*. If $x \leq y$ and $y \leq x$, we write $x \cong y$ and say x and y are *equivalent*. We call a set with a preorder a *poset*.

We have already introduced a few examples of posets using Hasse diagrams. It will be convenient to continue to do this, so let us be a bit more formal about what we mean. First, we need to define a graph.

Definition 1.22. A *graph* $G = (V, A, s, t)$ consists of a set V whose elements are called *vertices*, a set A whose elements are called *arrows*, and two functions $s, t: A \rightarrow V$ known as the *source* and *target* functions respectively. Given $a \in A$ with $s(a) = v$ and $t(a) = w$, we say that a is an arrow from v to w .

Remark 1.23. From every graph we can get a poset. Indeed, Hasse diagram is a graph $G = (V, A, s, t)$ that gives a *presentation* of a poset (P, \leq) . The elements of P are the vertices V in G , and the order \leq is given by $v \leq w$ iff there is a path from $v \rightarrow w$. By a path in G we mean any sequence of arrows such that the target of one arrow is the source of the next. This includes sequences of length 1, which are just arrows A in G , and sequences of length 0, which just start and end at the same vertex, without traversing any arrows. Thus for any vertex v is always a path $v \rightarrow v$, and this translates into the reflexivity law from Definition 1.21. The fact that paths $u \rightarrow v$ and $v \rightarrow w$ can be concatenated to a path $u \rightarrow w$ translates into the transitivity law.

Remark 1.24 (Partial orders are skeletal posets). A preorder is a partial order if we additionally have that

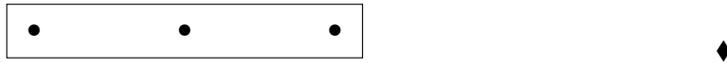
3. $x \cong y$ implies $x = y$.

Contrary to the definition we've chosen, the term poset frequently is used to mean *partially* ordered set, rather than preordered set. In category theory terminology, the requirement that $x \cong y$ implies $x = y$ is known as *skeletality*. We thus call partially ordered sets *skeletal posets*.

The difference between preorders and partial orders is rather minor. We will not spell out the details, but every poset can be considered equivalent to a skeletal poset.

Example 1.25 (Discrete posets). Every set X can be considered as a discrete poset. This means that the only order relations on X are of the form $x \leq x$; if $x \neq y$ then neither $x \leq y$ or $y \leq x$ hold.

We depict discrete posets as simply a collection of points:



Exercise 1.26. Does a collection of points, like the one in [Example 1.25](#), count as a Hasse diagram? ♦

Example 1.27 (Booleans). The booleans $\mathbb{B} = \{\text{false}, \text{true}\}$ form a preorder with $\text{false} \leq \text{true}$.



Exercise 1.28. Let X be the set of partitions of $\{\bullet, \circ, *\}$; it has five elements and an order by coarseness, as shown in the Hasse diagram [Eq. \(1.2\)](#). Write down every pair (x, y) of elements in X such that $x \leq y$. There should be 12. ♦

Example 1.29 (Natural numbers). The natural numbers \mathbb{N} are a poset with the order given by the usual size ordering, e.g. $0 \leq 1$ and $5 \leq 100$. We call this a linear or total order, because its Hasse diagram looks like a line:

$$\overset{0}{\bullet} \longrightarrow \overset{1}{\bullet} \longrightarrow \overset{2}{\bullet} \longrightarrow \overset{3}{\bullet} \longrightarrow \dots$$

The formal way to define *total order* is: for every two elements a, b , either $a \leq b$ or $b \leq a$. This holds for natural numbers. What made [Eq. \(1.2\)](#) not look like a line is that there are non-comparable elements (a, b) —namely all those in the middle row—which satisfy neither $a \leq b$ nor $b \leq a$.

Note that for any set S , there are many different ways of assigning an order to S . Indeed, for the set \mathbb{N} , we could also use the discrete ordering: only write $n \leq m$ if $n = m$. Another ordering is the reverse ordering, like $5 \leq 3$ and $3 \leq 2$, like how golf is scored (5 is worse than 3).

Yet another ordering on \mathbb{N} is given by division: we say that $n \leq m$ if n divides into m without remainder. In this ordering $2 \leq 4$, for example, but $2 \not\leq 3$, since there is a remainder when 2 is divided into 3. ♦

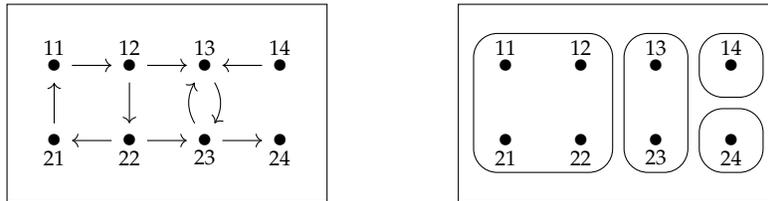
Exercise 1.30. Write down the numbers $1, 2, \dots, 10$ and draw an arrow $a \rightarrow b$ if a divides perfectly into b . Is it a total order? \diamond

Example 1.31 (Real numbers). The real numbers \mathbb{R} are also a poset with the order given by the usual size ordering. \blacklozenge

Exercise 1.32. Is the usual \leq ordering on the set \mathbb{R} of real numbers a total order? \diamond

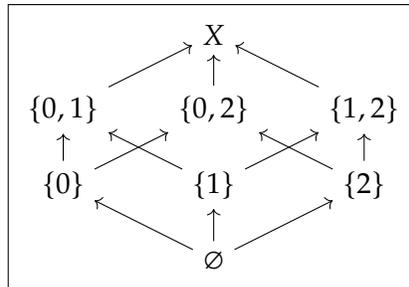
Example 1.33 (Partition from preorder). Given a poset, i.e. a pre-ordered set (P, \leq) , we defined the notion of equivalence of elements, denoted $x \cong y$, to mean $x \leq y$ and $y \leq x$. This is an equivalence relation, so it induces a partition on P .²

For example, the poset whose Hasse diagram is drawn on the left corresponds to the partition drawn on the right.



Example 1.34 (Power set). Given a set X , the set of subsets of X is known as the *power set* of X ; we denote it PX . The power set can be given an order by inclusion of subsets.

For example, taking $X = \{0, 1, 2\}$, we depict PX as



The Hasse diagram for the power set of a finite set, say $P\{1, 2, \dots, n\}$, always looks like a cube of dimension n . \blacklozenge

Exercise 1.35. Draw the Hasse diagrams for $P(\emptyset)$, $P\{1\}$, and $P\{1, 2\}$. \diamond

Example 1.36 (Partitions). Given a set A , we may also consider the poset of *partitions* of A . As we discussed in Proposition 1.11, partitions of A can be identified with equivalence relations on A .

The set of partitions on A form a poset, which we denote $\mathcal{E}(A)$. The order on partitions is given by fineness: a partition P is finer than a partition Q if, for every part $p \in P$ there is a part $q \in Q$ such that $A_p \subseteq A_q$. This is the order we gave in Eq. (1.2).

Recall from Remark 1.16 that partitions on A can be thought of as surjective functions out of A . Then $f: A \twoheadrightarrow P$ is finer than $g: A \twoheadrightarrow Q$ if there is a function $h: P \rightarrow Q$ such that $f.h = g$. \blacklozenge

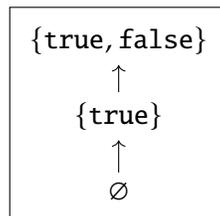
²The phrase “ A induces B ” means that we have an automatic way to turn an A into a B .

Exercise 1.37. For any set S there is a coarsest partition, having just one part. What surjective function does it correspond to?

There is also a finest partition, where everything is in its own partition. What surjective function does it correspond to? \diamond

Example 1.38 (Upper sets). Given a preorder (P, \leq) , an *upper set* in P is a subset U of P satisfying the condition that if $p \in U$ and $p \leq q$, then $q \in U$. “If p is an element then so is anything bigger.” Write $\mathcal{U}P$ for the set of upper sets in P . We can give the set \mathcal{U} an order by letting $U \leq V$ if U is contained in V .

For example, if (\mathbb{B}, \leq) is the booleans (Example 1.27), then its poset of upper sets $\mathcal{U}\mathbb{B}$ is

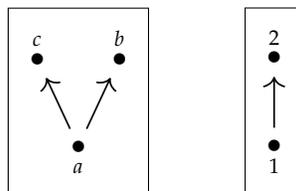


Just $\{\text{false}\}$ by itself is not an upper set, because $\text{false} \leq \text{true}$. \diamond

Exercise 1.39. Prove that the poset of upper sets on a discrete poset (see Example 1.25) on a set X is simply the power set PX . \diamond

Example 1.40 (Product poset). Given posets (P, \leq) and (Q, \leq) , we may define a poset structure on the product set $P \times Q$ by setting $(p, q) \leq (p', q')$ if and only if $p \leq p'$ and $q \leq q'$. We call this the *product poset*. This is a basic example of a more general construction known as the product of categories. \diamond

Exercise 1.41. Draw the product of the two posets drawn below:



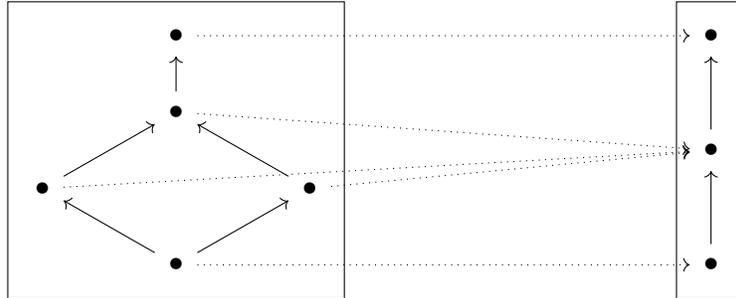
Example 1.42 (Opposite poset). Given a poset (P, \leq) , we may define the opposite poset (P, \leq^{op}) to have the same set of elements, but with $p \leq^{\text{op}} q$ if and only if $q \leq p$. \diamond

1.3 Monotone maps

We have said that the categorical perspective emphasizes relationships between things. For example, posets are settings in which we have one sort of relationship, \leq , and any two objects may be, or may not be, so-related. Jumping up a level, the categorical perspective emphasizes that posets themselves can be related. The most important sort of relationship between posets is called a *monotone map*. These are functions that preserve poset orders, and hence the right notion of structure-preserving map for posets.

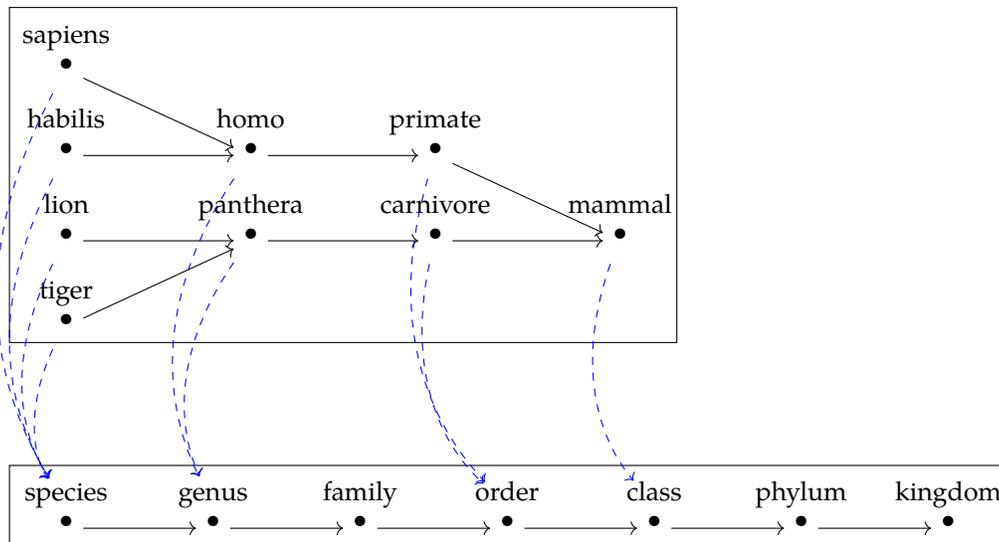
Definition 1.43. A *monotone map* between posets (A, \leq_A) and (B, \leq_B) is a function $f: A \rightarrow B$ such that, for all elements $x, y \in A$, if $x \leq_A y$ then $f(x) \leq_B f(y)$.

A monotone map between two posets associates an element of the second poset to every element of the first. We depict this by drawing a dotted arrow from each element x of the first poset to the relevant element $f(x)$ of the second poset. Note that the order is preserved, so if one element x is above another y in the poset on the left, then the image $f(x)$ of the first is above the image $f(y)$ of the second in the poset on the right.



Example 1.44. Let \mathbb{B} be the poset of booleans from Example 1.27. The map $\mathbb{B} \rightarrow \mathbb{N}$ sending false to 17 and true to 24 is a monotone map. \blacklozenge

Example 1.45 (The tree of life). Consider the set of all animal classifications, for example ‘tiger’, ‘mammal’, ‘sapiens’, ‘mammal’, etc.. These are ordered by specificity: since ‘tiger’ is a type of ‘mammal’, we write $\text{tiger} \leq \text{mammal}$. The result is a poset, which in fact forms a tree, often called the tree of life. At the top of the following diagram we see a small part of it:



At the bottom we see the hierarchical structure as a poset. The dashed arrows show a monotone map, call it F , from the classifications to the hierarchy. It is monotone because whenever there is a path $x \rightarrow y$ upstairs, there is a path $F(x) \rightarrow F(y)$ downstairs. \blacklozenge

Example 1.46. Given a finite set X , the map $\mathcal{P}(X) \rightarrow \mathbb{N}$ sending each subset S to its number of elements $\#S$ is monotone map. \blacklozenge

Example 1.47. Given a poset (P, \leq) , the map $\mathcal{U}(P) \rightarrow \mathcal{P}(P)$ mapping each upper set of (P, \leq) to itself as a subset of P is a monotone map. \blacklozenge

Exercise 1.48. Show that monotone maps between discrete posets are just functions. \blacklozenge

Example 1.49. Recall from Example 1.36 that given a set X we define $\mathcal{E}X$ to be the set of partitions on X , and that a partition may be defined using a surjective function $s: X \twoheadrightarrow P$ for some set P .

Any surjective function $f: X \rightarrow Y$ induces a monotone map $f^*: \mathcal{E}Y \rightarrow \mathcal{E}X$, going “backwards”. It is defined by sending a partition $s: Y \twoheadrightarrow P$ to the composite $f.s: X \twoheadrightarrow P$.³ \blacklozenge

Exercise 1.50. Choose two sets X and Y with at least three elements each and choose a surjective, non-identity function $f: X \rightarrow Y$ between them. Write down two different partitions P and Q of Y , and then find f^*P and f^*Q . \blacklozenge

The following proposition, 1.51, is straightforward to check. Recall the definition of the identity function from Example 1.14 and the definition of composition from Definition 1.19.

Proposition 1.51. *For any poset (P, \leq_P) , the identity function is monotone.*

If (Q, \leq_Q) and (R, \leq_R) are posets and $f: P \rightarrow Q$ and $g: Q \rightarrow R$ are monotone, then $(f.g): P \rightarrow R$ is also monotone.

Exercise 1.52. Check the two claims made in Proposition 1.51. \blacklozenge

Example 1.53. Recall the definition of opposite poset from Example 1.42. The identity function $\text{id}_P: P \rightarrow P$ is a monotone map $(P, \leq) \rightarrow (P, \leq^{\text{op}})$ if and only if for all $p, q \in P$ we have $q \leq p$ whenever $p \leq q$. For historical reasons connected to linear algebra, when this is true, we call (P, \leq) a *dagger poset*. Since this says that $p \leq q$ if and only if $q \leq p$, in fact a dagger poset is a set with a transitive, reflexive, and symmetric binary relation, and hence a set equipped with an equivalence relation (Definition 1.10). \blacklozenge

Exercise 1.54. Recall the skeletal posets (Remark 1.24) and discrete posets (Example 1.25). Show that a skeletal dagger poset is just a discrete poset, and hence just a set. \blacklozenge

Definition 1.55. Let (P, \leq_P) and (Q, \leq_Q) be posets. A monotone function $f: P \rightarrow Q$ is called an *isomorphism* if there exists a monotone function $g: Q \rightarrow P$ such that $f.g = \text{id}_P$ and $g.f = \text{id}_Q$. This means that for any $p \in P$ and $q \in Q$, we have

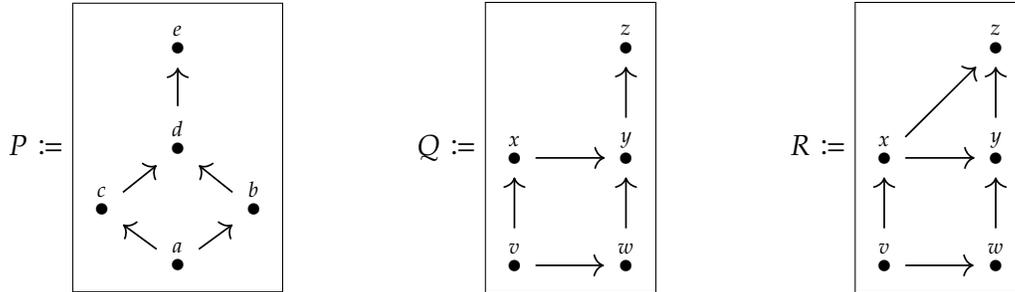
$$p = g(f(p)) \quad \text{and} \quad q = f(g(q)).$$

If there is an isomorphism $P \rightarrow Q$, we say that P and Q are *isomorphic*.

³We will later see that any function f induces a monotone map $\mathcal{E}Y \rightarrow \mathcal{E}X$, but it involves an extra step: replacing the composite $f.s$ with its image. See Section 1.5.2.

An isomorphism between posets is basically just a relabeling the elements.

Example 1.56. Here are the Hasse diagrams for three posets P , Q , and R , all of which are isomorphic:



The map $f: P \rightarrow Q$ given by $f(a) = v$, $f(b) = w$, $f(c) = x$, $f(d) = y$, and $f(e) = z$ has an inverse.

In fact Q and R are the same poset. One may be confused by the fact that there is an arrow $x \rightarrow z$ in the Hasse diagram for R and not one in Q , but in fact this arrow is superfluous. By the transitivity property of posets (Definition 1.21): since $x \leq y$ and $y \leq z$, we must have $x \leq z$, whether it is drawn or not. In the same way, we could have drawn an arrow $v \rightarrow y$ and it would not have changed the poset. \blacklozenge

Recall the poset $\mathbb{B} = \{\text{false}, \text{true}\}$, where $\text{false} \leq \text{true}$. As simple as this poset is, it is also one of the most important.

Exercise 1.57. Show that the map Φ from the introduction, which was roughly given by ‘Is \bullet connected to $*$?’ is a monotone map from the poset shown in Eq. (1.2) to \mathbb{B} . \blacklozenge

Proposition 1.58. *Let P be a poset. Monotone maps $P \rightarrow \mathbb{B}$ are in one-to-one correspondence with upper sets of P .*

Proof. Let $f: P \rightarrow \mathbb{B}$ be a monotone map. Consider $f^{-1}(\text{true})$. Note that if $p \in f^{-1}(\text{true})$ and $p \leq q$, then $\text{true} = f(p) \leq f(q)$. But true is a maximal element of \mathbb{B} , so $\text{true} \leq f(q)$ implies $\text{true} = f(q)$, which implies $q \in f^{-1}(\text{true})$. This shows that $f^{-1}(\text{true})$ is an upper set.

Conversely, if U is an upper set in P , define $f_U: P \rightarrow \mathbb{B}$ such that $f_U(p) = \text{true}$ when $p \in U$, and $f_U(p) = \text{false}$ when $p \notin U$. This is a monotone map, because if $p \leq q$, then either $p \in U$, so $q \in U$ and $f(p) = \text{true} = f(q)$, or $p \notin U$, so $f(p) = \text{false} \leq f(q)$.

These two constructions are mutually inverse, and hence prove the proposition. \square

Exercise 1.59 (Pullback map). Let P and Q be posets, and $f: P \rightarrow Q$ be a monotone map. Then we can define a monotone map $f^*: \mathcal{U}Q \rightarrow \mathcal{U}P$ sending an upper set $U \subseteq Q$ to the upper set $f^{-1}(U) \subseteq P$. We call this the *pullback along f* .

Viewing upper sets as a monotone maps to \mathbb{B} as in Proposition 1.58, the pullback can be understood in terms of composition. Indeed, show that the f^* is defined by taking $u: Q \rightarrow \mathbb{B}$ to $f.u: P \rightarrow \mathbb{B}$. \blacklozenge

1.4 Meets and joins

As we have said, posets are sets with a simple order relationship between the elements. We can use this structure to define and construct elements with special properties. We have discussed joins before, but we discuss them again now that we have built up some formalism.

1.4.1 Definition and basic examples

Consider the poset (\mathbb{R}, \leq) of real numbers ordered in the usual way. The subset $\mathbb{N} \subseteq \mathbb{R}$ has many lower bounds, namely $r < 0$ is a lower bound. It also has a greatest lower bound—also called a meet—which is 0. It does not have any upper bound. The set $\{\frac{1}{n+1} \mid n \in \mathbb{N}\} = \{1, \frac{1}{2}, \frac{1}{3}, \dots\}$ has both a greatest lower bound, namely 0, and a least upper bound, namely 1.

These notions will have correlates in category theory, called limits and colimits, which we will discuss in the Chapter 3. For now, we want to make the definition of greatest lower bounds and least upper bounds, called meets and joins, precise.

Definition 1.60. Let (P, \leq) be a poset, and let $A \subseteq P$ be a subset. We say that an element $p \in P$ is the *meet* of A if

1. for all $a \in A$, we have $p \leq a$, and
2. for all q such that $q \leq a$ for all $a \in A$, we have that $q \leq p$.

We write $p = \bigwedge A$, or $p = \bigwedge_{a \in A} a$. If A just consists of two elements, say $A = \{a, b\}$, we can denote $\bigwedge A$ simply by $a \wedge b$.

Similarly, we say that p is the *join* of A if

1. for all $a \in A$ we have $a \leq p$, and
2. for all q such that $a \leq q$ for all $a \in A$, we have that $p \leq q$.

We write $p = \bigvee A$ or $p = \bigvee_{a \in A} a$, or when $A = \{a, b\}$ we may simply write $p = a \vee b$.

Note that a subset A need not have a meet or a join in an arbitrary poset (P, \leq) .

Example 1.61. Consider the two-element set $P = \{p, q, r\}$ with the discrete ordering. The set $A = \{p, q\}$ does not have a join in P because if x was a join, we would need $p \leq x$ and $q \leq x$, and there is no such element x . ♦

Example 1.62. In any poset P , we have $p \vee p = p \wedge p = p$. ♦

Example 1.63. In a power set, the meet of a collection of subsets is their intersection, while the join is their union. This justifies the terminology. ♦

Example 1.64. In a total order, the meet of a set is its infimum, while the join of a set is its supremum. ♦

Exercise 1.65. Recall the division ordering on \mathbb{N} from Example 1.29: we say that $n \leq m$ if n divides perfectly into m . What is the meet of two numbers in this poset? What about the join? ♦

Proposition 1.66. *Suppose (P, \leq) is a poset and $A \subseteq B \subseteq P$ are subsets that have meets. Then $\bigwedge B \leq \bigwedge A$.*

Similarly, if A and B have joins, then $\bigvee A \leq \bigvee B$.

Proof. Let $m = \bigwedge A$ and $n = \bigwedge B$. Then for any $a \in A$ we also have $a \in B$, so $n \leq a$. Thus n is a lower bound for A and hence $n \leq m$, because m is the greatest lower bound. The second claim is proved similarly. \square

Exercise 1.67. Let (P, \leq) be a poset and $p \in P$ an element. Consider the set $A = \{p\}$ with one element. Show that $\bigwedge A = p$.

Is the same true when \bigwedge is replaced by \bigvee ? \diamond

1.4.2 Back to observations and generative effects

In [Ada17], Adam thinks of monotone maps as observations. A monotone map $\Phi: P \rightarrow Q$ is a phenomenon of P as observed by Q . He defines generative effects of such a map Φ to be its failure to preserve joins (or more generally, for categories, its failure to preserve colimits).

Definition 1.68. We say that a monotone map $\phi: P \rightarrow Q$ *preserves meets* if $\phi(a \wedge b) = \phi(a) \wedge \phi(b)$ for all $a, b \in P$. We similarly say ϕ *preserves joins* if $\phi(a \vee b) = \phi(a) \vee \phi(b)$ for all $a, b \in P$.

Definition 1.69. We say that a monotone map $\phi: P \rightarrow Q$ *sustains generative effects* if there exist elements $a, b \in P$ such that

$$\phi(a) \vee \phi(b) \neq \phi(a \vee b).$$

If we think of ϕ as an observation or measurement of the systems a and b , then the left hand side may be interpreted as the combination of the observation of a with the observation of b . On the other hand, the right hand side is our observation of the combined system $a \vee b$. The inequality implies that the we see something when we observe the combined system that we could not expect by merely combining our observations of the pieces. That is, that there are generative effects from the interconnection of the two systems.

In his work on generative effects, Adam restricts his attention to maps that preserve meets, even while they do not preserve joins. The preservation of meets implies that the map ϕ behaves well when restricting to a subsystem, even if it can throw up surprises when joining systems.

This discussion naturally leads into Galois connections, which are pairs of monotone maps between posets, one of which preserves all joins and the other of which preserves all meets.

1.5 Galois connections

The preservation of meets and joins, and hence whether a monotone map sustains generative effects, is tightly related to the concept of a Galois connection, or more generally an adjunction.

1.5.1 Definition and examples of Galois connections

Galois connections between posets were first considered by Évariste Galois—who didn't call them by that name—in the context of a connection he found between “field extensions” and “automorphism groups”. We will not discuss this further, but the idea is that given two posets P and Q , a Galois connection is a pair of maps back and forth—from P to Q and from Q to P —with certain properties, which make it like a relaxed version of isomorphisms. To be a bit more precise, isomorphisms are examples of Galois connections, but Galois connections need not be isomorphisms.

Definition 1.70. A *Galois connection* between posets P and Q is a pair of monotone maps $f: P \rightarrow Q$ and $g: Q \rightarrow P$ such that

$$f(p) \leq q \quad \text{if and only if} \quad p \leq g(q). \tag{1.5}$$

We say that f is the *left adjoint* and g is the *right adjoint*.

Example 1.71. Consider the map $(3 \times -): \mathbb{N} \rightarrow \mathbb{N}$ which sends $x \in \mathbb{N}$ to $3x$. Write $\lceil z \rceil$ for the smallest natural number greater than $z \in \mathbb{R}$, and write $\lfloor z \rfloor$ for the largest natural number smaller than $z \in \mathbb{R}$, e.g. $\lceil 3.14 \rceil = 4$ and $\lfloor 3.14 \rfloor = 3$.

It is easily checked that

$$x \leq 3y \text{ if and only if } \lceil x/3 \rceil \leq y.$$

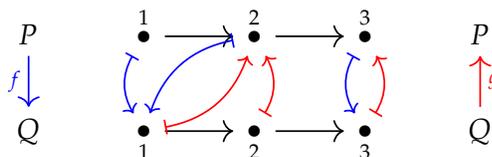
Thus we have a Galois connection between $\lceil -/3 \rceil$ and $3 \times -$. Similarly, it is easily checked that

$$3x \leq y \text{ if and only if } x \leq \lfloor y/3 \rfloor.$$

Thus $\lfloor -/3 \rfloor$ is right adjoint to $3 \times -$. ♦

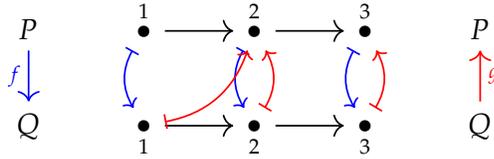
Exercise 1.72. Consider the poset $P = Q = \underline{3}$.

1. Let f, g be the monotone maps shown below:



Is it the case that f is left adjoint to g ? Check that for each $1 \leq p, q \leq 3$, one has $f(p) \leq q$ iff $p \leq g(q)$.

2. Let f, g be the monotone maps shown below:



Is it the case that f is left adjoint to g ? ◇

Remark 1.73. If P and Q are total orders and $f: P \rightarrow Q$ and $g: Q \rightarrow P$ are drawn with arrows bending as in Exercise 1.72, we believe that f is left adjoint to g iff the arrows do not cross. But we have not proved this, mainly because it is difficult to state precisely, and the total order case is not particularly general.

Example 1.74. Here is an attempt to create a biological example; it is meant to be correct when taken as a vague idea, not to stand up to serious scrutiny.

Let (P, \subseteq) denote the poset of possible animal populations: an element p is a set of possible animals, and we write $p \subseteq q$ if every animal in p is also in q . Let (G, \leq) denote the set of gene pools: an element g is again a set of animals, but we write $g \leq h$ if the gene pool h can generate the gene pool g , i.e. if every animal in g could be generated by mating animals in h (genes recombine, no mutations).

There is a monotone map $i: P \rightarrow G$ sending each population to itself as a gene pool, $i(p) = p$, because if $p \subseteq q$ then clearly q can generate p . There is also a monotone map $\text{cl}: G \rightarrow P$ sending each gene pool to its closure: the set of all possible animals that could be generated from g under recombination of genes.

These are adjoint: given $p \in P$ and $g \in G$, then $p \subseteq \text{cl}(g)$ means that every animal in p can be obtained by recombination of genetic material in g , which is exactly what $p \leq g$ means. Since $p = i(p)$, we have the desired Eq. (1.5): $i(p) \leq g$ iff $p \subseteq \text{cl}(g)$. ◇

Exercise 1.75. Does $\lfloor -/3 \rfloor$ have a right adjoint $R: \mathbb{N} \rightarrow \mathbb{N}$? If not, why? If so, does its right adjoint have a right adjoint? ◇

1.5.2 Back to partitions

Recall from Example 1.36 that we can understand partitions on a set S in terms of surjective functions out of S .

Suppose given any function $g: S \rightarrow T$. We will show that this function g induces a Galois connection between poset of S -partitions and the poset of T -partitions. The way you would explain it to a seasoned category theorist is:

The left adjoint is given by taking any surjection out of S and pushing out along g to get a surjection out of T . The right adjoint is given by taking any surjection out of T , composing with g to get a function out of S , and then

taking the epi-mono factorization to get a surjection out of S .

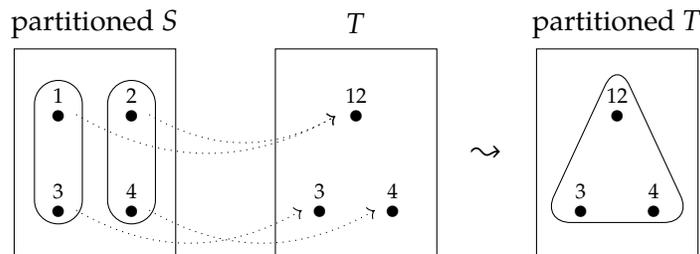


By the end of this book, the reader will understand pushouts and epi-mono factorizations, so she will be able to make sense of the above statement. But for now we will explain the process in more down-to-earth terms.

Start with $g: S \rightarrow T$; we first want to understand $g_!: \mathcal{E}S \rightarrow \mathcal{E}T$. So start with a partition of S . To begin the process of obtaining a partition on T , say two elements $t_1, t_2 \in T$ are in the same part, $t_1 \sim t_2$, if there exist $s_1, s_2 \in S$ that are in the same part, $s_1 \sim s_2$, such that $g(s_1) = t_1$ and $g(s_2) = t_2$. However, the result of doing so will not necessarily be transitive—you may get $t_1 \sim t_2$ and $t_2 \sim t_3$ without $t_1 \sim t_3$ —and partitions must be transitive. So complete the process by just adding in the missing pieces (take the transitive closure).

Again starting with g , we want to get the right adjoint $g^*: \mathcal{E}T \rightarrow \mathcal{E}S$. So start with a partition of T . Get a partition on S by saying that two elements $s_1, s_2 \in S$ are in the same part $s_1 \sim s_2$ iff $g(s_1) \sim g(s_2)$ in T .

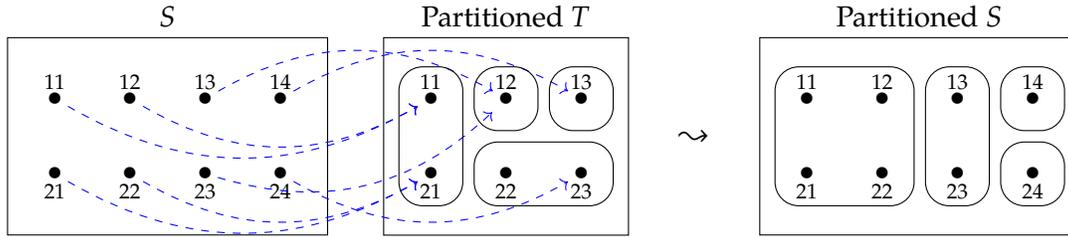
Example 1.76. Let $S = \{1, 2, 3, 4\}$, $T = \{12, 3, 4\}$, and $g: S \rightarrow T$ by $g(1) = g(2) = 12$, $g(3) = 3$, and $g(4) = 4$. The partition shown left below is translated by $g_!$ to the partition shown on the right.



Exercise 1.77. There are 15 different partitions of a set with four elements. Choose 6 different ones and for each one $c: S \twoheadrightarrow P$, find $g_!(c)$, where S, T , and $g: S \rightarrow T$ are the same as they were in Example 1.76. ♦

Example 1.78. Let S, T be as below, and let $g: S \rightarrow T$ be the function shown in blue. Here is a picture of how g^* takes a partition on T and turns “pulls it back” to a partition

on S :



◆

Exercise 1.79. There are five partitions possible on a set with three elements, say $T = \{12, 3, 4\}$. Using the same S and $g: S \rightarrow T$ as in Example 1.76, determine the partition $g^*(c)$ on S for each of the five partitions $c: T \rightarrow P$. ◆

To check that, for any function $g: S \rightarrow T$, the monotone map $g_!: \mathcal{E}S \rightarrow \mathcal{E}T$ really is left adjoint to $g^*: \mathcal{E}T \rightarrow \mathcal{E}S$ would take too much time for this sketch. But the following exercise gives some evidence.

Exercise 1.80. Let S, T , and $g: S \rightarrow T$ be as in Exercise 1.77.

1. Choose a nontrivial partition $c: S \rightarrow P$ and let $g_!(c)$ be its push forward partition on T .
2. Choose any coarser partition $d: T \rightarrow P'$, i.e. where $g_!(c) \leq d$.
3. Choose any non-coarser partition $e: T \rightarrow Q$, i.e. where $g_!(c) \not\leq e$. (If you can't do this, revise your answer for #1.)
4. Find $g^*(d)$ and $g^*(e)$.
5. The adjunction formula Eq. (1.5) in this case says that since $g_!(c) \leq d$ and $g_!(c) \not\leq e$, we should have $c \leq g^*(d)$ and $c \not\leq g^*(e)$. Show that this is true.

◆

1.5.3 Basic theory of Galois connections

Proposition 1.81. Suppose that $f: P \rightarrow Q$ and $g: Q \rightarrow P$ are functions. The following are equivalent

- a.) f and g form a Galois connection where f is left adjoint to g ,
- b.) for every $p \in P$ and $q \in Q$ we have

$$p \leq g(f(p)) \quad \text{and} \quad f(g(q)) \leq q. \quad (1.6)$$

Proof. Suppose f is left adjoint to g . Take any $p \in P$, and let $q = f(p)$. By reflexivity, we have $f(p) \leq q$, so by Definition 1.70 we have $p \leq g(q)$, but this means $p \leq g(f(p))$. The proof that $f(g(q)) \leq q$ is similar.

Now suppose that Eq. (1.6) holds for all $p \in P$ and $q \in Q$. We want show that $f(p) \leq q$ iff $p \leq g(q)$. Suppose $f(p) \leq q$; then since g is monotonic, $g(f(p)) \leq g(q)$, but $p \leq g(f(p))$ so $p \leq g(q)$. The proof that $p \leq g(q)$ implies $f(p) \leq q$ is similar. □

Exercise 1.82. Complete the proof of Proposition 1.81 by showing that

1. if f is left adjoint to g then for any $q \in Q$, we have $f(g(q)) \leq q$
2. if Eq. (1.6) holds, then for any $p \in P$ and $q \in Q$, if $p \leq g(q)$ then $f(p) \leq q$.

◇

If we replace \leq with $=$ in Proposition 1.81, we get back the definition of isomorphism (Definition 1.55); this is why we said at the beginning of Section 1.5.1 that Galois connections are a kind of relaxed version of isomorphisms.

Exercise 1.83. Show that if $f: P \rightarrow Q$ has a right adjoint g , then it is unique up to isomorphism. That means, for any other right adjoint g' , we have $g(q) \cong g'(q)$ for all $q \in Q$.

Is the same true for left adjoints? That is, if $h: P \rightarrow Q$ has a left adjoint, is it necessarily unique? ◇

Proposition 1.84 (Right adjoints preserve meets). *Let $f: P \rightarrow Q$ be left adjoint to $g: Q \rightarrow P$. Suppose $A \subseteq Q$ any subset, and let $g(A) = \{g(a) \mid a \in A\}$ be its image. Then if A has a meet $\bigwedge A \in Q$ then $g(A)$ has a meet $\bigwedge g(A)$ in P , and we have*

$$g\left(\bigwedge A\right) = \bigwedge g(A).$$

Similarly, left adjoints preserve joins. That is, if $A \subseteq P$ is any subset that has a join $\bigvee A \in P$, then $f(A)$ has a join $\bigvee f(A)$ in Q , and we have

$$f\left(\bigvee A\right) = \bigvee f(A).$$

Proof. Let $f: P \rightarrow Q$ and $g: Q \rightarrow P$ be adjoint monotone maps, with g right adjoint to f . Let $A \subseteq Q$ be any subset and let $m := \bigwedge A$ be its meet. Then since g is monotone $g(m) \leq g(a)$ for all $a \in A$, so $g(m)$ is a lower bound for the set $g(A)$. We will be done if we can show it is the greatest lower bound.

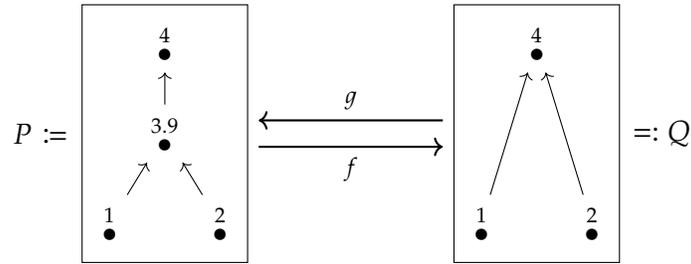
So take any other lower bound b for $g(A)$; that is suppose that for all $a \in A$, we have $b \leq g(a)$. Then by definition of g being a right adjoint (Definition 1.70), we also have $f(b) \leq a$. This means that $f(b)$ is a lower bound for A in Q . Since the meet m is the greatest lower bound, we have $f(b) \leq m$. Once again using the Galois connection, $b \leq g(m)$, proving that $g(m)$ is indeed the greatest lower bound for $g(A)$, as desired.

The second claim is proved similarly; see Exercise 1.85. □

Exercise 1.85. Complete the proof of Proposition 1.84 by showing that left adjoints preserve joins. ◇

Since left adjoints preserve joins, we know that they cannot sustain generative effects.

Example 1.86. Right adjoints need not preserve joins. Here is an example:



Let g be the map that preserves labels, and let f be the map that preserves labels as far as possible but sends $f(3.9) = 4$. Both are monotonic, and one can check that g is right adjoint to f (see Exercise 1.87). But g does not preserve joins because $1 \vee 2 = 4$ holds in Q , whereas $1 \vee 2 = 3.9$ in P . \blacklozenge

Exercise 1.87. To be sure that g really is right adjoint to f in Example 1.86, there are twelve things to check; do so. That is, for every $p \in P$ and $q \in Q$, check that $f(p) \leq q$ iff $p \leq g(q)$. \blacklozenge

Thus Galois connections provide a way of constructing maps that sustain generative effects: maps that preserve meets but do not necessarily preserve joins. In fact, all maps that preserve meets come from Galois connections as we now show.

Proposition 1.88 (Adjoint functor theorem for posets). *Suppose Q is a poset that has all meets and let P be any poset. A monotone map $g: Q \rightarrow P$ preserves meets if and only if it is a right adjoint.*

Similarly, if P has all joins and Q is any poset, a monotone map $f: P \rightarrow Q$ preserves joins if and only if it is a left adjoint.

Proof. We will only prove the claim about meets; the claim about joins follows similarly.

We proved in Proposition 1.84 that right adjoints preserve meets. Suppose that g is a monotone map that preserves meets; we shall construct a left adjoint f . We define our candidate $f: P \rightarrow Q$ on any $p \in P$ by

$$f(p) := \bigwedge \{q \in Q \mid p \leq g(q)\}; \quad (1.7)$$

this meet is well defined because Q has all meets, but for f to really be a candidate, we need to show it is monotone. So suppose that $p \leq p'$. Then $\{q' \in Q \mid p' \leq g(q')\} \subseteq \{q \in Q \mid p \leq g(q)\}$. By Proposition 1.66, this implies $f(p) \leq f(p')$. Thus f is monotone.

It remains to prove the adjointness condition: given $p_0 \in P$ and $q_0 \in Q$, we want to show that $p_0 \leq g(q_0)$ if and only if $f(p_0) \leq q_0$. Suppose first $f(p_0) \leq q_0$. Then since g is monotonic, $g(f(p_0)) \leq g(q_0)$; we have $p_0 \leq g(f(p_0))$ by Proposition 1.81, so this implies $p_0 \leq g(q_0)$.

For the other direction, suppose $p_0 \leq g(q_0)$. Then $\{q_0\} \subseteq \{q \in Q \mid p_0 \leq g(q)\}$, so take the meet of both sides. Again using Proposition 1.66, the definition (1.7) of $f(p_0)$, and the fact that $\bigwedge \{q_0\} = q_0$ (see Exercise 1.67), we get $f(p_0) \leq q_0$, as desired. \square

Example 1.89. Let $f: X \rightarrow Y$ be a function between sets. We can imagine X as a set of balls, Y as a set of buckets, and f as putting each ball in a bucket.

Then we have the monotone map $f^*: PY \rightarrow PX$ that category theorists call “pullback along f ”. This map takes a subset $B \subseteq Y$ to its preimage $f^{-1}B \subseteq X$: that is, it takes a collection B of buckets, and tells you all the balls that lie in them. This operation is monotonic (more buckets means more balls) and it has both a left and a right adjoint.

The left adjoint $f_!(A)$ is given by the direct image: it maps a subset $A \subseteq X$ to

$$f_!(A) := \{y \in Y \mid \text{there exists } a \in A \text{ such that } fa = y\}$$

This map hence takes a set A of balls, and tells you all the buckets that contain at least one of these balls.

The right adjoint f_* maps a subset $A \subseteq X$ to

$$f_*(A) := \{y \in Y \mid \text{for all } a \text{ such that } fa = y \text{ we have } a \in A\}$$

This map takes a set A of balls, and tells you all the buckets that only contain balls from A .

Notice that all three of these operations turn out to be interesting: start with a set B of buckets and return all the balls in them, or start with a set A of balls and either find the buckets that contain at least one ball from A , or the buckets whose only balls are from A . But we did not invent these mappings f^* , $f_!$, and f_* : they were *induced* by the function f . They were automatic. It is one of the pleasures of category theory that adjoints so often turn out to have interesting semantic interpretations. \blacklozenge

Exercise 1.90. Choose sets X and Y with between two and four elements each, and choose a function $f: X \rightarrow Y$.

1. Choose two different subsets $B_1, B_2 \subseteq Y$ and find $f^{-1}(B_1)$ and $f^{-1}(B_2)$.
2. Choose two different subsets $A_1, A_2 \subseteq X$ and find $f_!(A_1)$ and $f_!(A_2)$.
3. With the same $A_1, A_2 \subseteq X$, find $f_*(A_1)$ and $f_*(A_2)$. \blacklozenge

1.5.4 Closure operators

Given a Galois connection with $f: P \rightarrow Q$ left adjoint to $g: Q \rightarrow P$, we may compose f and g to arrive at a monotone map $f.g: P \rightarrow P$ from a poset to itself. This monotone map has the property that $p \leq (f.g)(p)$, and that $(f.g.f.g)(p) \cong (f.g)(p)$ for any $p \in P$. This is an example of a *closure operator*.⁴

Exercise 1.91. Suppose that if f is left adjoint to g . Use Proposition 1.81 to show the following.

1. $p \leq (f.g)(p)$
2. $(f.g.f.g)(p) \cong (f.g)(p)$. To prove this, show inequalities in both directions, \leq and \geq . \blacklozenge

⁴The other composite $g.f$ satisfies the dual properties: $(g.f)(q) \leq q$ and $(g.f.g.f)(q) \cong (g.f)(q)$ for all $q \in Q$. This is called an *interior operator*, though we will not discuss this concept further.

Definition 1.92. A closure operator $j: P \rightarrow P$ on a preorder P is a monotone map such that for all $p \in P$ we have

- $p \leq j(p)$;
- $j(j(p)) \cong j(p)$.

Example 1.93. An example of closure operators comes from computation. Imagine computation as a process of rewriting input expressions to output expressions. For example, a computer can rewrite the expression $7+2+3$ as the expression 12 . The set of arithmetic expressions has a partial order according to whether one expression can be rewritten as another.

We might think of a computer program, then, as a method of taking an expression and reducing it to another expression. So it is a map $j: \text{exp} \rightarrow \text{exp}$. It furthermore is desirable to require that this computer program is a closure operator. Monotonicity means that if an expression x can be rewritten into expression y , then the evaluation $j(x)$ can be rewritten into $j(y)$. Moreover, the requirement $x \leq j(x)$ implies that j can only turn one expression into another if doing so is a permissible rewrite. The requirement $j(j(x)) = j(x)$ implies if you feed in an expression that has already been reduced, the computer program leaves it as is. These properties provide useful structure in the analysis of program semantics. \blacklozenge

Example 1.94 (Adjunctions from closure operators). Just as every adjunction gives rise to a closure operator, from every closure operator we may construct an adjunction. Let P be a poset and let j be a closure operator on P . We can define a poset fix_j to have elements the fixed points of j ; that is,

$$\text{fix}_j := \{p \in P \mid j(p) = p\}.$$

This is a subset of P , and inherits an order as a result. Note that, since j is a closure operator, $j(j(p)) = j(p)$, so $j(p)$ is a fixed point for all $p \in P$.

We define an adjunction with left adjoint $j: P \rightarrow \text{fix}_j$ sending p to $j(p)$, and right adjoint $g: \text{fix}_j \rightarrow P$ simply the inclusion of sub-posets. To see it's really an adjunction, we need to see that for any $p \in P$ and $q \in \text{fix}_j$, we have $f(p) \leq q$ if and only if $p \leq q$. Let's check it. Since $p \leq j(p)$, we have that $j(p) \leq q$ implies $p \leq q$ by transitivity. Conversely, since q is a fixed point, $p \leq q$ implies $j(p) \leq j(q) = q$. \blacklozenge

Example 1.95. Another example of closure operators comes from logic. This will be discussed in the final chapter of the book, in particular Section 7.4.5, but basically logic is the study of when one formal statement—or proposition—implies another. For example, if n is prime then n is not a multiple of 6, or if it is raining then the ground is getting wetter. Here “ n is prime”, “ n is not a multiple of 6”, “it is raining”, and “the ground is getting wetter” are propositions, and we gave two implications.

Take the set of all propositions, and order them by $p \leq q$ iff p implies q , denoted $p \Rightarrow q$. Since $p \Rightarrow p$ and since whenever $p \Rightarrow q$ and $q \Rightarrow r$, we also have $p \Rightarrow r$, this is indeed a poset.

A closure operator is often called a *modal operator*. It is a function j from propositions to propositions, where $p \Rightarrow j(p)$ and $j(j(p)) = j(p)$. An example of a j is “assuming Bob is in San Diego...”. Think of this as a proposition B ; so “assuming Bob is in San Diego, p ” means $B \Rightarrow p$. Let’s see why $B \Rightarrow -$ is a closure operator.

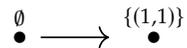
If ‘ p ’ is true then ‘assuming Bob is in San Diego, p ’ is still true. Suppose that ‘assuming Bob is in San Diego it is the case that, assuming Bob is in San Diego, p ’ is true. It follows that ‘assuming Bob is in San Diego, p ’ is true. So we have seen, at least informally, that “assuming Bob is in San Diego...” is a closure operator. \blacklozenge

1.5.5 Level shifting

A phenomenon that happens often in category theory is something we might call level-shifting. It is easier to give an example of this than to explain it directly.

Given any set S , there is a set $\mathbf{Rel}(S)$ of binary relations on S . An element $R \in \mathbf{Rel}(S)$ is formally a subset $R \subseteq S \times S$. The set $\mathbf{Rel}(S)$ can be given an order: say that $R \leq R'$ if $R \subseteq R'$, i.e. if whenever $R(s_1, s_2)$ holds then so does $R'(s_1, s_2)$.

For example, the Hasse diagram for $\mathbf{Rel}(\{1\})$ is:



Exercise 1.96. Draw the Hasse diagram for the poset $\mathbf{Rel}(\{1, 2\})$ of all binary relations on the set $\{1, 2\}$ \blacklozenge

For any set S , there is also a set $\mathbf{Pos}(S)$, consisting of all the poset relations on S . In fact there is a poset structure \leq on $\mathbf{Pos}(S)$, again given by inclusion: \leq is below \leq' ($\leq \leq \leq'$) if $a \leq b$ implies $a \leq' b$ for every $a, b \in S$. A poset of poset structures? That’s what we mean by a level shift.

Every poset relation is—in particular—a relation, so we have an inclusion $\mathbf{Pos}(S) \rightarrow \mathbf{Rel}(S)$. This is actually the right adjoint of a Galois connection. Its right adjoint is a monotone map $\text{Cl}: \mathbf{Rel}(S) \rightarrow \mathbf{Pos}(S)$ given by taking any relation and taking the reflexive and transitive closure, i.e. adding $s \leq s$ for every s and adding $s \leq u$ whenever $s \leq t$ and $t \leq u$.

Exercise 1.97. Let $S = \{1, 2, 3\}$.

1. Come up with any preorder relation \leq on S , and let $L \subseteq S \times S$ be the set $\{(s_1, s_2) \mid s_1 \leq s_2\}$, i.e. L is the image of \leq under the inclusion $\mathbf{Pos}(S) \rightarrow \mathbf{Rel}(S)$.
2. Come up with any two binary relations $Q \subseteq S \times S$ and $Q' \subseteq S \times S$ such that $L \leq Q$ but $L \not\leq Q'$.
3. Show that $\leq \leq \text{Cl}(Q)$.
4. Show that $\leq \not\leq \text{Cl}(Q')$. \blacklozenge

1.6 Summary and further reading

In this first chapter, we set the stage for category theory by introducing one of the simplest interesting type of example: posets. From this seemingly simple structure, a bunch of further structure emerges: monotone maps, meets, joins. In terms of modeling real world phenomena, we thought of posets as the states of a system, and monotone maps as describing a way to use one system to observe another. From this point of view, generative effects occur when our way of observing a system does not preserve joins, so looking at the parts separately and combining that information does not tell the story of the whole system.

In the final section we introduced Galois connections. These are a relaxation of the idea of a pair of inverse map, called instead a pair of adjoint maps, with the notion of *almost* inverse being described by the orders. Perhaps surprisingly, it turns out these are closely related to joins and meets: monotone maps have adjoints if and only if they preserve meets or joins.

The next two chapters build significantly on this material, but in two different directions. Chapter 2 adds a new operation on the underlying set: it introduces the idea of a monoidal structure on posets. This allows us to construct an element $a \otimes b$ of a poset P from any elements $a, b \in P$, in a way that respects the order structure. On the other hand, Chapter 3 adds new structure on the order itself: it introduces the idea of a morphism, which describes not only whether $a \leq b$, but gives a name f for how a relates to b . This structure is known as a category. Nonetheless, but generalizations are fundamental to the story of compositionality, and in Chapter 4 we'll see them meet in the concept of a monoidal category. The lessons we have learned in this category will illuminate their more complicated generalizations in the chapters to come. Indeed, it is a useful principle in studying category theory to try to understand concepts first in the setting of posets, where often much of the complexity is stripped away and one can develop some intuition before considering the general case.

But perhaps you might be interested in exploring some ideas in this chapter in other directions. While we won't return to them in this book, we learned about generative effects from Elie Adam's thesis [Ada17], and a much richer treatment of generative effect can be found there. In particular, he introduces the notion of an abelian category and of cohomology, which provides a way to detect the potential for a system, when combined with others, to produce generative effects.

Another important application of posets, monotone maps, and Galois connections is to the analysis of programming languages. In this setting, posets describe the possible states of a computer, and monotone maps describe the action of programs, or relationships between different ways of modeling computation states. Galois connections are useful for showing different models are closely related, and that reasoning can be transported from one framework to another. For more detail on this, see Chapter 4 of the textbook [NNH99].

Resource theories: Monoidal posets and enrichment

2.1 Getting from a to b

You can't make an omelette without breaking an egg. To obtain the things we want requires resources, and the process of transforming from what we have into what we want is often an intricate one. In this chapter, we will discuss how monoidal posets can help us think about this matter.

Consider the following three questions you might ask yourself:

- Given what I have, is it *possible* to get what I want?
- Given what I have, what is the *minimum cost* to get what I want?
- Given what I have, what is the *set of ways* to get what I want?

These questions are about resources—those you have and those you want—but perhaps more importantly, they are about moving from have to want: possibility of, cost of, and ways to.

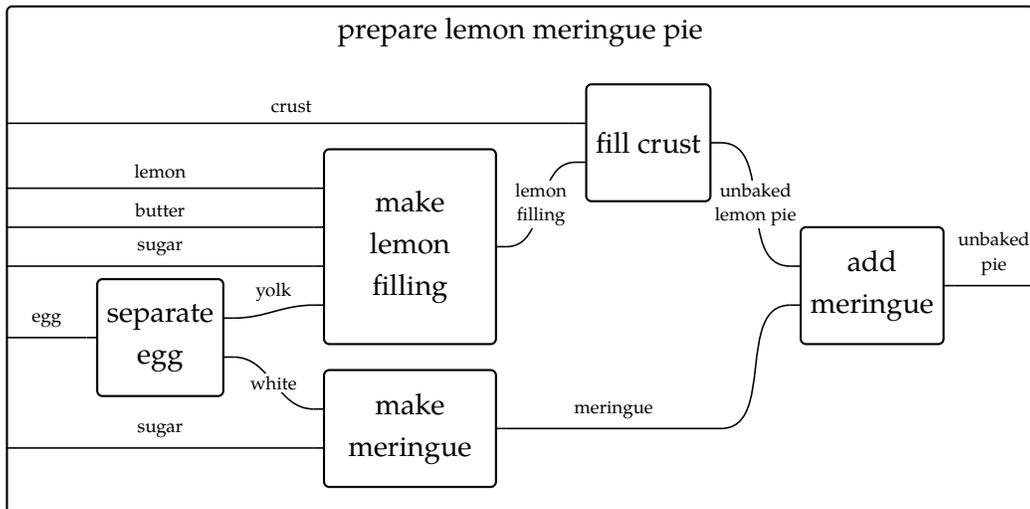
Such questions come up not only in our lives, but also in science and industry. In chemistry, one asks whether a certain set of compounds can be transformed into another set, how much energy such a reaction will require, or what plans exist for making it happen. In manufacturing, one asks similar questions.

From an external point of view, both a chemist and an industrial firm might be regarded as store-houses of information on the above subjects. The chemist knows which compounds she can make given other ones, and how to do so; the firm has stored knowledge of the same sort. The research work of the chemist and the firm is to use what they know in order to derive—or discover—new knowledge.

This is roughly the first goal of this chapter: to discuss a formalism for expressing recipes—how one set of resources can be transformed into another—and how to derive new recipes from old. The idea here is not complicated, neither in life nor in our mathematical formalism. The value added then is to simply see how it works, so we

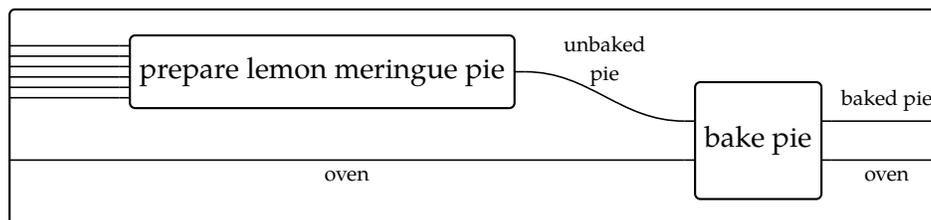
can build on it within the book, and so others can build on it in their own work.

We briefly discuss the categorical approach to this idea—namely that of *monoidal posets*—for building new recipes from old. The following *wiring diagram* shows, assuming one knows how to implement each of the interior boxes, how to implement the preparation of a lemon meringue pie:



(2.1)

The wires show resources: we start with crust, lemon, butter, egg, and sugar resources, and we end up with an unbaked pie resource. We could take this whole method and combine it with others, e.g. baking the pie:



In the above example we see that resources are not always consumed when they are used. For example, we use an oven to convert—or catalyze the transformation of—an unbaked pie into a baked pie, and we get the oven back after we are done. To use economic terms, some resources are more durable than others.

String diagrams are an important mathematical structure that will come up repeatedly in this book. They were invented in the mathematical context—more specifically in the context of monoidal categories—by Joyal and Street [JS93], but they have been used less formally by engineers and scientists in various contexts for a long time.

As we said above, our first goal in this chapter is to use monoidal posets, and the corresponding wiring diagrams, as a language for building new recipes from old. Our second goal is to discuss something called \mathcal{V} -categories for various monoidal posets \mathcal{V} . A \mathcal{V} -category is a set of objects, which one may think of as points on a map, and \mathcal{V} structures the question of getting from point a to point b .

The examples of monoidal posets \mathcal{V} that we will be most interested in are called **Bool** and **Cost**. Roughly speaking, a **Bool**-category is a set of points where the question of getting from point a to point b has a true / false answer. A **Cost**-category is a set of points where the question of getting from a to b has an answer $d \in [0, \infty]$, a cost. objects such that the measure of relationship between any two is a number $r \in [0, \infty]$. Once we add some rules about how these various measurements compose, we will have a \mathcal{V} -category.

This story works in more generality than monoidal posets. Indeed, in Chapter 4 we will discuss something called a monoidal category, which generalizes monoidal posets, and generalize the definition of \mathcal{V} -category accordingly. In this more general setting, \mathcal{V} -categories can also address our third question above, describing *sets* of ways of getting from a to b .

We will begin in Section 2.2 by defining symmetric monoidal posets, giving a few preliminary examples, and discussing wiring diagrams. We then give many more examples of symmetric monoidal posets, including both some real-world examples in the form of resource theories and some mathematical examples that will come up again throughout the book. Next, in Section 2.3 we discuss enrichment and \mathcal{V} -categories—how monoidal posets structure the question of getting from a to b —before talking about important constructions on \mathcal{V} -categories (Section 2.4), and using matrix multiplication to analyze \mathcal{V} -categories (Section 2.5).

2.2 Symmetric monoidal posets

In Section 1.2 we introduced posets. The notation for a poset, namely (X, \leq) , refers to two pieces of structure: a set called X and a relation called \leq that is reflexive and transitive.

We want to add to the concept of posets a way of combining elements in X , taking two elements and adding or multiplying them together. However, the operation does not have to literally be addition or multiplication; it only needs to satisfy some of the properties one expects from them.

2.2.1 Definition and first examples

We begin with a formal definition of symmetric monoidal posets.

Definition 2.1. A *symmetric monoidal structure* on a poset (X, \leq) consists of two constituents:

- (i) an element $I \in X$, called the *monoidal unit*, and
- (ii) a function $\otimes: X \times X \rightarrow X$, called the *monoidal product*.

These constituents must satisfy the following properties:

- (a) for all $x_1, x_2, y_1, y_2 \in X$, if $x_1 \leq y_1$ and $x_2 \leq y_2$, then $x_1 \otimes x_2 \leq y_1 \otimes y_2$,
- (b) for all $x \in X$, the equations $I \otimes x = x$ and $x \otimes I = x$ hold,

- (c) for all $x, y, z \in X$, the equation $(x \otimes y) \otimes z = x \otimes (y \otimes z)$ holds, and
- (d) for all $x, y \in X$, the equivalence $x \otimes y \cong y \otimes x$ holds.

A poset equipped with a symmetric monoidal structure, (X, \leq, I, \otimes) , is called a *symmetric monoidal poset*.

Anyone can propose a set X , an order \leq on X , an element I in X , and a binary operation \otimes on X and ask whether (X, \leq, I, \otimes) is a symmetric monoidal poset. And it will indeed be one, as long as it satisfies rules a, b, c, and d of Definition 2.1.

Remark 2.2. It is often useful to replace $=$ with \cong throughout Definition 2.1. The result is a perfectly good notion, called a *weak monoidal structure*. The reason we chose equality is that it makes equations look simpler, which we hope aids first-time readers.

The notation for the monoidal unit and the monoidal product may vary: monoidal units we have seen include I (as in the definition), 0 , 1 , *true*, *false*, $\{*\}$, and more. Monoidal products we have seen include \otimes (as in the definition), $+$, $*$, \wedge , \vee , and \times . The *preferred notation* a given setting is whatever best helps our brains remember what we're trying to do; the names I and \otimes are just defaults.

Example 2.3. There is a well-known poset structure, denoted \leq , on the set \mathbb{R} of real numbers; e.g. $-5 \leq \sqrt{2}$. We propose 0 as a monoidal unit and $+$: $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ as a monoidal product. Does $(\mathbb{R}, \leq, 0, +)$ satisfy the conditions of Definition 2.1?

If $x_1 \leq y_1$ and $x_2 \leq y_2$, it is true that $x_1 + x_2 \leq y_1 + y_2$. It is also true that $0 + x = x$ and $x + 0 = x$, that $(x + y) + z = x + (y + z)$, and that $x + y = y + x$. Thus $(\mathbb{R}, \leq, 0, +)$ satisfies the conditions of being a symmetric monoidal poset. \blacklozenge

Exercise 2.4. Consider again the poset (\mathbb{R}, \leq) from Example 2.3. Someone proposes 1 as a monoidal unit and $*$ (usual multiplication) as a monoidal product. But an expert walks by and says “that won't work.” Figure out why, or prove the expert wrong! \blacklozenge

Example 2.5. Here is a non-example for people who know the game “standard poker”. Let H be the set of all poker hands, where a hand means a choice of five cards from the standard 52-card deck. As an order, put $h \leq h'$ if h' beats h in poker.

One could propose a monoidal product $\otimes: H \times H \rightarrow H$ by assigning $h_1 \otimes h_2$ to be “the best hand one can form out of the ten cards in h_1 and h_2 ”. If some cards are in both h_1 and h_2 , just throw them away. So for example $\{2\heartsuit, 3\heartsuit, 4\heartsuit, 6\spadesuit, 7\spadesuit\} \otimes \{2\heartsuit, 5\heartsuit, 6\heartsuit, 6\spadesuit, 7\spadesuit\} = \{2\heartsuit, 3\heartsuit, 4\heartsuit, 5\heartsuit, 6\heartsuit\}$, because the latter is the best hand you can make with the former two.

This proposal for a monoidal structure will fail the condition a. of Definition 2.1: it could be the case that $h_1 \leq i_1$ and $h_2 \leq i_2$, and yet *not* be the case that $h_1 \otimes h_2 \leq i_1 \otimes i_2$. For example, consider this case:

$$\begin{aligned} h_1 &:= \{2\heartsuit, 3\heartsuit, 10\spadesuit, J\spadesuit, Q\spadesuit\} & i_1 &:= \{4\clubsuit, 4\spadesuit, 6\heartsuit, 6\diamondsuit, 10\diamondsuit\} \\ h_2 &:= \{2\diamondsuit, 3\diamondsuit, 4\diamondsuit, K\spadesuit, A\spadesuit\} & i_2 &:= \{5\spadesuit, 5\heartsuit, 7\heartsuit, J\diamondsuit, Q\diamondsuit\}. \end{aligned}$$

Here, $h_1 \leq i_1$ and $h_2 \leq i_2$, but $h_1 \otimes h_2 = \{10\spadesuit, J\spadesuit, Q\spadesuit, K\spadesuit, A\spadesuit\}$ is the best possible hand and beats $i_1 \otimes i_2 = \{5\spadesuit, 5\heartsuit, 6\heartsuit, 6\diamondsuit, Q\diamondsuit\}$. \blacklozenge

answer is that a monoidal poset (X, \leq, I, \otimes) has some notion of element, relationship, and combination, and so do wiring diagrams: the wires represent elements, the boxes represent relationships, and the wiring diagrams themselves show how relationships can be combined. We call boxes and wires *icons*; we will see several more icons in this chapter, and throughout the book.

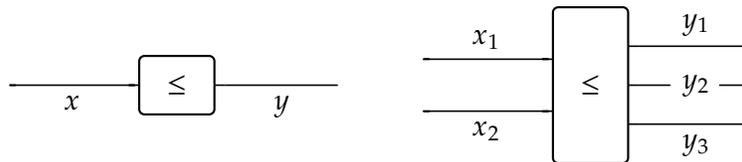
To get a bit more rigorous about the connection, let's start with a monoidal poset (X, \leq, I, \otimes) as in Definition 2.1. Each element $x \in X$ can be made the label of a wire. Given two objects x, y , we can either draw two wires in parallel—one labeled x and one labeled y —or we can draw one wire labeled $x \otimes y$; we consider those to be the same:

$$\left. \begin{array}{c} \text{---} \\ x \end{array} \quad \begin{array}{c} \text{---} \\ y \end{array} \quad \begin{array}{c} x \\ \text{---} \\ y \end{array} \right\} = \begin{array}{c} \text{---} \\ x \otimes y \end{array}$$

The monoidal unit I can either be drawn as a wire labeled I or as an absence of wires:

$$\begin{array}{c} \text{---} \\ I \end{array} = \textit{nothing}$$

The inequality $x \leq y$ is drawn as a box with a wire labeled x on the left and a wire labeled y on the right; see the first box below:

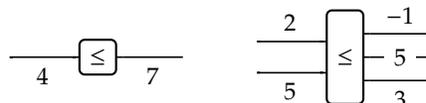


The second box corresponds to the inequality $x_1 \otimes x_2 \leq y_1 \otimes y_2 \otimes y_3$. Before going on to the properties from Definition 2.1, let us pause for an example of what we've discussed so far.

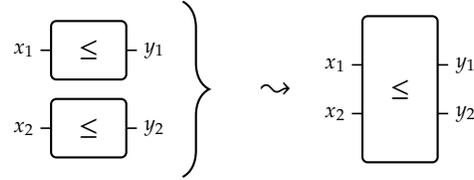
Example 2.6. Recall the symmetric monoidal poset $(\mathbb{R}, \leq, 0, +)$ from Example 2.3. The wiring diagrams for it allow wires labeled by real numbers. Drawing wires in parallel corresponds to adding their labels, and the wire labeled 0 is equivalent to no wires at all.

$$\left. \begin{array}{c} \text{---} \\ 3.14 \end{array} \quad \begin{array}{c} \text{---} \\ -1 \end{array} \quad \begin{array}{c} 3.14 \\ \text{---} \\ -1 \end{array} \right\} = \begin{array}{c} \text{---} \\ 2.14 \end{array} \quad \begin{array}{c} \text{---} \\ 0 \end{array} = \textit{nothing}$$

And here are a couple facts about $(\mathbb{R}, \leq, 0, +)$: $4 \leq 7$ and $2 + 5 \leq -1 + 5 + 3$.



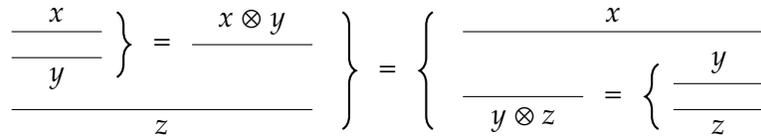
We now return to how the properties of monoidal posets (conditions (a)–(d) from Definition 2.1) correspond to properties of this sort of wiring diagram. Property (a) says that if $x_1 \leq y_1$ and $x_2 \leq y_2$ then $x_1 \otimes x_2 \leq y_1 \otimes y_2$. This corresponds to the idea that we may stack any two boxes in parallel and get a new box:



Condition (b), that $I \otimes x = x$ and $x \otimes I = x$, says the following:



Condition (c), that $(x \otimes y) \otimes z = x \otimes (y \otimes z)$ says the following:



But this looks much harder than it is: the associative property should be thought of as saying that the stuff on the very left above equals the stuff on the very right, i.e.

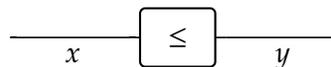


Finally, the symmetry condition (d), that $x \otimes y = y \otimes x$, says that wires are allowed to cross:

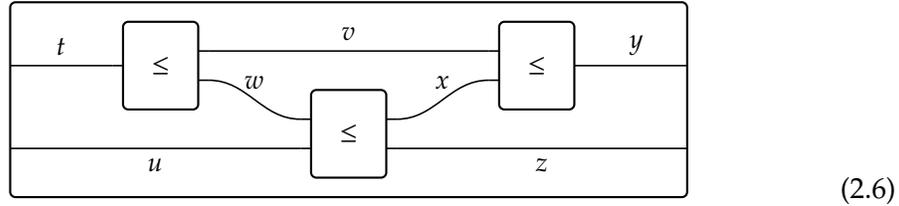


One may regard the pair of crossing wires as another icon in our iconography, in addition to the boxes and wires we already have.

Wiring diagrams as graphical proofs Given a monoidal poset $\mathcal{X} = (X, \leq, I, \otimes)$, a wiring diagram is a graphical proof of something about \mathcal{X} . Each box in the diagram has a left side and a right side, say x and y , and represents the assertion that $x \leq y$.



A wiring diagram is a bunch of interior boxes connected together inside an exterior box. It represents a graphical proof that says: if all of the interior assertions are correct, then so is the exterior assertion.



The inner boxes in Eq. (2.6) translate into the assertions:

$$t \leq v + w \quad w + u \leq x + z \quad v + x \leq y \quad (2.7)$$

and the outer box translates into the assertion:

$$t + u \leq y + z. \quad (2.8)$$

The whole wiring diagram 2.6 says “if you know that the assertions in 2.7 are true, then I am a proof that the assertion in 2.8 is also true.” What exactly is the proof that Eq. (2.6) represents?

$$t + u \leq v + w + u \leq v + x + z \leq y + z. \quad (2.9)$$

Example 2.7. Recall the lemon meringue pie wiring diagram from Eq. (2.1). It has five interior boxes, such as “separate egg” and “fill crust”, and it has one exterior box called “prepare lemon meringue pie”. Each box is the assertion that, given the collection of resources on the left, say an egg, you can transform it into the collection of resources on the right, say an egg white and an egg yolk. The whole string diagram is a proof that if each of the interior assertions is true—i.e. you really do know how to separate eggs, make lemon filling, make meringue, fill crust, and add meringue—then the exterior assertion is true: you can prepare a lemon meringue pie. \blacklozenge

Exercise 2.8. The string of inequalities in Eq. (2.9) is not quite a proof, because technically there is no such thing as $v+w+u$, for example. Instead, there is $(v+w)+u$ and $v+(w+u)$, and so on.

1. Formally prove, using only the rules of symmetric monoidal posets (Definition 2.1), that given the assertions in Eq. (2.7), the conclusion in Eq. (2.8) follows.
2. Reflexivity and transitivity should show up in your proof. Make sure you are explicit about where they do.
3. How can you look at the wiring diagram Eq. (2.6) and know that the commutative axiom (Definition 2.1 d.) does not need to be invoked?

\blacklozenge

We next discuss some examples of symmetric monoidal posets. We begin in Section 2.2.3 with some more concrete examples, from science, commerce, and informatics. Then in Section 2.2.4 we discuss some examples arising from pure math, some of which will get a good deal of use later on, e.g. in Chapter 4.

2.2.3 Applied examples

Resource theories are studies of how resources are exchanged in a given arena. For example, in social resource theory one studies a marketplace where combinations of goods can be traded for—as well as converted into—other combinations of goods.

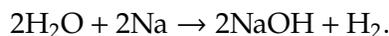
Whereas marketplaces are very dynamic, and an apple might be tradable for an orange on Sunday but not on Monday, what we mean by resource theory in this chapter is a static notion: deciding “what buys what,” once and for all.¹ This sort of static notion of conversion might occur in chemistry: the chemical reactions that take place one day will quite likely take place a different day as well. Manufacturing may be somewhere in between: the set of production techniques—whereby a company can convert one set of resources into another—do not change much from day to day.

We learned about resource theories from [CFS16; Fri17], who go much further than we will; see Section 2.6 for more information. In this section we will focus only on the main idea. While there are many beautiful mathematical examples of symmetric monoidal posets, as we will see in Section 2.2.4, there are also ad hoc examples coming from life experience. In the next chapter, on databases, we will see the same theme: while there are some beautiful mathematical categories out there, database schemas are *ad hoc* organizational patterns of information. Describing something as “ad hoc” is often considered derogatory, but it just means “formed, arranged, or done for a particular purpose only”. There is nothing wrong with doing things for a purpose; it’s common outside of pure math and pure art. Let’s get to it.

Chemistry In high school chemistry, we work with chemical equations, where material collections such as



are put together in the form of reaction equations, such as



The collection on the left, $2\text{H}_2\text{O} + 2\text{Na}$ is called the *reactant*, and the collection on the right, $2\text{NaOH} + \text{H}_2$ is called the *product*.

We can consider reaction equations such as the one above as taking place inside a single symmetric monoidal poset $(\text{Mat}, \rightarrow, 0, +)$. Here Mat is the set of all collections of atoms and molecules, sometimes called *materials*. So we have $\text{NaCl} \in \text{Mat}$ and $4\text{H}_2\text{O} + 6\text{Ne} \in \text{Mat}$.

The set Mat has a poset structure denoted by the \rightarrow symbol, which is the preferred symbol in the setting of chemistry. To be clear, \rightarrow is taking the place of the order relation \leq from Definition 2.1. The $+$ symbol is the preferred notation for the monoidal

¹Using some sort of temporal theory, e.g. the one presented in Chapter 7, one could take the notion here and have it change in time.

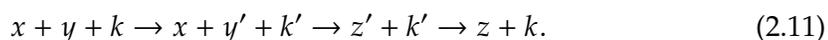
product in the chemistry setting, taking the place of \otimes . While it does not come up in practice, we use 0 to denote the monoidal unit.

Exercise 2.9. Here is an exercise for people familiar with reaction equations: check that conditions (a), (b), (c), and (d) of Definition 2.1 hold. \diamond

An important notion in chemistry is that of catalysis: one compound *catalyzes* a certain reaction. For example, one might have the following set of reactions:

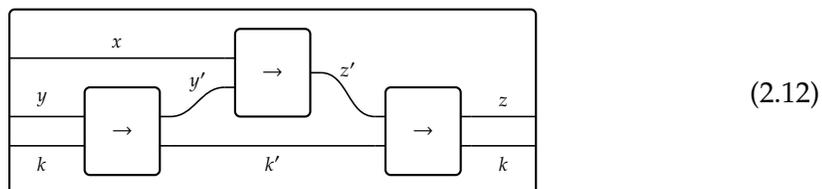


Using the laws of monoidal posets, we obtain the composed reaction



Here k is the catalyst because it is found both in the reactant and the product of the reaction. It is said to catalyze the reaction $x + y \rightarrow z$. The idea is that the reaction $x + y \rightarrow z$ cannot take place given the reactions in Eq. (2.10). But if k is present, meaning if we add k to both sides, the resulting reaction can take place.

The wiring diagram for the reaction in Eq. (2.11) is shown in Eq. (2.12). The three interior boxes correspond to the three reactions given in Eq. (2.10), and the exterior box corresponds to the composite reaction $x + y + k \rightarrow z + k$.



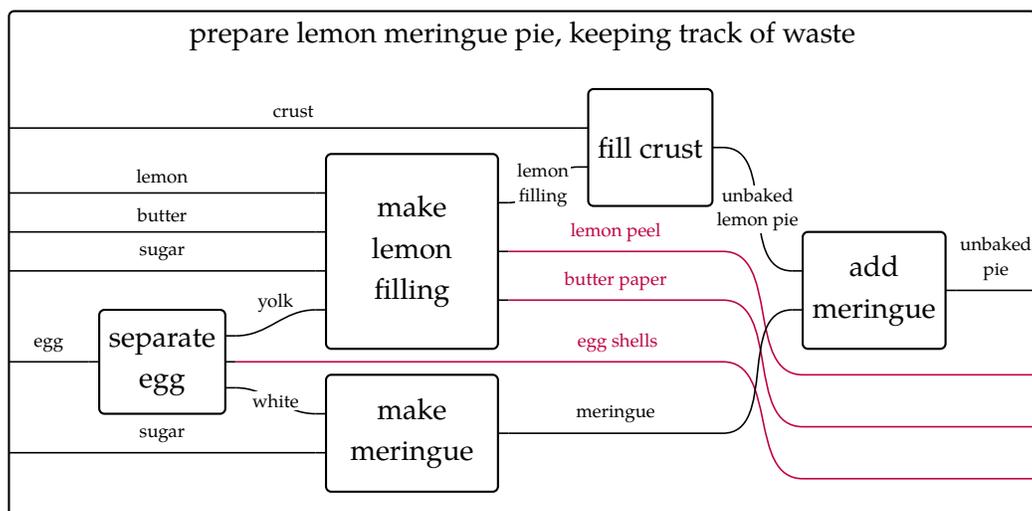
Manufacturing Whether we are talking about baking pies, building smart phones, or following pharmaceutical recipes, manufacturing firms need to store basic recipes, and build new recipes by combining simpler recipes in schemes like the one shown in Eq. (2.1) or Eq. (2.12).

The basic idea in manufacturing is exactly the same as that for chemistry, except there is an important assumption we can make in manufacturing that does not hold for chemical reactions:

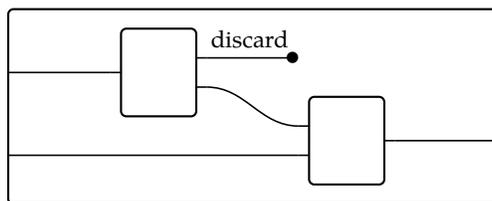
You can throw anything you want away, and it disappears from view.

This simple assumption has caused the world some significant problems, but it is still in effect. In our meringue pie example, we can ask: “what happened to the egg shell, or the paper surrounding the stick of butter”? The answer is they were thrown away. It would certainly clutter our diagram and our thinking if we had to carry these resources

through the diagram:



Instead, in our daily lives and in manufacturing, we do not have to hold on to something if we don't need it; we can just discard it. In terms of wiring diagrams, this can be shown using a new icon $\text{---}\bullet$, as follows:



(2.13)

To model this concept of waste using monoidal categories, one just adds an additional axiom to (a), (b), (c), and (d) from Definition 2.1:

$$(e) \quad x \leq I \text{ for all } x \in X. \quad (\text{discard axiom})$$

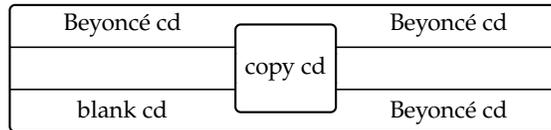
It says that every x can be converted into the monoidal unit I . In the notation of the chemistry section, we would write instead $x \rightarrow 0$: any x yields nothing. But this is certainly not accepted in the chemistry setting. For example,



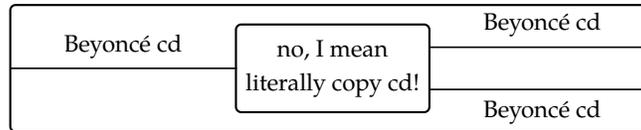
is certainly not a legal chemical equation. It is easy to throw things away in manufacturing, because we assume that we have access to each item produced. In chemistry, when you have 10^{23} of substance A dissolved in something else, you cannot just simply discard A . So axiom (e) is valid in manufacturing but not chemistry.

Recall that in Section 2.2.2 we said that there were many different styles of wiring diagrams. Now we're saying that adding the discard axiom changes the wiring diagram style, in that it adds this new discard icon that allows wires to terminate, as shown in Eq. (2.13). In informatics, we will change the wiring diagram style yet again.

Informatics A major difference between information and a physical object is that information can be copied. Whereas one cup of butter never becomes two, it is easy for a single email to be sent to two different people. It is much easier to copy a music file than it is to copy a CD. Here we do not mean “copy the information from one compact disc onto another”—of course that’s easy—instead, we mean that it’s quite difficult to copy the physical disc, thereby forming a second physical disc! In diagrams, the distinction is between the relation



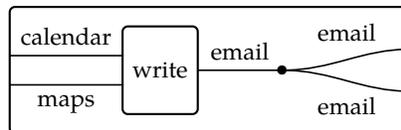
and the relation



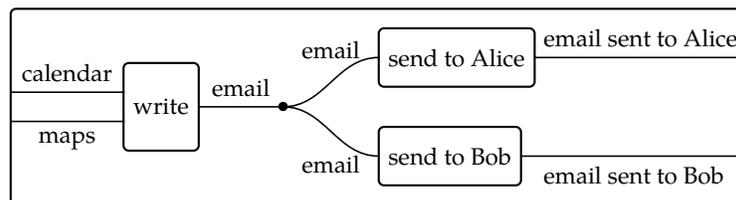
The former is possible, the latter is magic.

Of course material objects can sometimes be copied; cell mitosis is a case in point. But this is a remarkable biological process, certainly not something that is expected for ordinary material objects. In the physical world, we would make mitosis a box transforming one cell into two. But in (classical, not quantum) information, everything can be copied, so we add a new icon to our repertoire.

Namely, in wiring diagram notation, copying information appears as a new icon, \leftarrow , allowing us to split wires:



Now with two copies of the email, we can send one to Alice and one to Bob.



(2.14)

Information can also be discarded, at least in the conventional way of thinking, so in addition to axioms (a) to (d) from Definition 2.1, we can keep axiom (e) from 39 and add a new *copy axiom*:

$$(f) \quad x \leq x + x \text{ for all } x \in X.$$

(copy axiom)

allowing us to make mathematical sense of diagrams like Eq. (2.14).

Now that we have examples of monoidal posets under our belts, let's discuss some nice mathematical examples.

2.2.4 Abstract examples

In this section we discuss several mathematical examples of symmetric monoidal structures on posets.

The Booleans The simplest nontrivial poset is the booleans: $\mathbb{B} = \{\text{true}, \text{false}\}$ with $\text{false} \leq \text{true}$. There are two different symmetric monoidal structures on it.

Example 2.10 (Booleans with AND). We can define a monoidal structure on \mathbb{B} by letting the monoidal unit be `true` and the monoidal product be \wedge (AND). If one thinks of `false` = 0 and `true` = 1, then \wedge corresponds to the usual multiplication operation $*$. That is, with this correspondence, the two tables below match up:

$$\begin{array}{c|cc} \wedge & \text{false} & \text{true} \\ \hline \text{false} & \text{false} & \text{false} \\ \text{true} & \text{false} & \text{true} \end{array} \qquad \begin{array}{c|cc} * & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array} \qquad (2.15)$$

One can check that all the properties in Definition 2.1 hold, so we have a monoidal poset which we denote $\mathbf{Bool} := (\mathbb{B}, \leq, \text{true}, \wedge)$. \blacklozenge

\mathbf{Bool} will be important when we get to the notion of enrichment. Enriching in a monoidal poset $\mathcal{V} = (V, \leq, I, \otimes)$ means “letting \mathcal{V} structure the question of getting from a to b ”. All of the structures of a monoidal poset —i.e. the set V , the ordering relation \leq , the monoidal unit I , and the monoidal product \otimes —play a role in how enrichment works.

For example, let's look at the case of $\mathbf{Bool} = (\mathbb{B}, \leq, \text{true}, \wedge)$. The fact that its underlying set is $\mathbb{B} = \{\text{false}, \text{true}\}$ will translate into saying that “getting from a to b is a true/false question”. The fact that `true` is the monoidal unit will translate into saying “you can always get from a to a ”. The fact that \wedge is the monoidal product will translate into saying “if you can get from a to b AND you can get from b to c then you can get from a to c ”. Finally, the “if-then” form of the previous sentence is coming from the order relation \leq . We will make this more precise in Section 2.3.

We will be able to play the same game with other monoidal posets, as we will see after we define a monoidal poset called \mathbf{Cost} in Example 2.19.

Some other monoidal posets It is a bit imprecise to call \mathbf{Bool} “the” boolean monoidal poset, because there is another monoidal structure on $(\mathbb{B}, \leq, \text{true})$, which we describe in Exercise 2.11. The first structure, however, seems to be more useful in practice than the second.

Exercise 2.11. Let (\mathbb{B}, \leq) be as above, but now consider the monoidal product to be \vee (OR).

\vee	false	true
false	false	true
true	true	true

max	0	1
0	0	1
1	1	1

What must the monoidal unit be in order to satisfy the conditions of Definition 2.1?

Does it satisfy the rest of the conditions? ◇

Here are two different monoidal structures on the poset (\mathbb{N}, \leq) of natural numbers, where \leq is the usual ordering ($0 \leq 1$ and $5 \leq 16$).

Example 2.12 (Natural numbers with addition). There is a monoidal structure on (\mathbb{N}, \leq) where the monoidal unit is 0 and the monoidal product is $+$, i.e. $6 + 4 = 10$. It is easy to check that $x_1 \leq y_1$ and $x_2 \leq y_2$ implies $x_1 + x_2 \leq y_1 + y_2$, as well as all the other conditions of Definition 2.1. ◇

Exercise 2.13. Show there is a monoidal structure on (\mathbb{N}, \leq) where the monoidal product is $*$, i.e. $6 * 4 = 24$. What should the monoidal unit be? ◇

Example 2.14 (Divisibility and multiplication). Recall from Example 1.29 that there is a “divisibility” order on \mathbb{N} : we write $m|n$ to mean that m divides into n without remainder. So $1|m$ for all m and $4|12$.

There is a monoidal structure on $(\mathbb{N}, |)$, where the monoidal unit is 1 and the monoidal product is $*$, i.e. $6 * 4 = 24$. Then if $x_1|y_1$ and $x_2|y_2$, then $(x_1 * x_2)|(y_1 * y_2)$. Indeed, if there is some $p_1, p_2 \in \mathbb{N}$ such that $p_1 = \frac{y_1}{x_1}$ and $p_2 = \frac{y_2}{x_2}$, then $\frac{y_1 * y_2}{x_1 * x_2} = p_1 * p_2$. ◇

Exercise 2.15. Again taking the divisibility order $(\mathbb{N}, |)$. Someone proposes 0 as the monoidal unit and $+$ as the monoidal product. Does that proposal satisfy the conditions of Definition 2.1? Why or why not? ◇

Exercise 2.16. Consider the poset (P, \leq) with Hasse diagram $\boxed{\text{no} \rightarrow \text{maybe} \rightarrow \text{yes}}$. We propose a monoidal structure with yes as the monoidal unit and “min” as the monoidal product.

1. Make sense of “min” by filling in the multiplication table with elements of P .

min	no	maybe	yes
no	?	?	?
maybe	?	?	?
yes	?	?	?

2. Check the axioms of Definition 2.1 hold for $\mathbf{NMY} := (P, \leq, \text{yes}, \text{min})$, given your definition of min. ◇

Exercise 2.17. Let S be a set and let $\mathbb{P}(S)$ be its powerset, the set of all subsets of S , including the empty subset, $\emptyset \subseteq S$, and the “everything” subset, $S \subseteq S$. We can

give $\mathbb{P}(S)$ an order: $A \leq B$ is given by the subset relation $A \subseteq B$, as discussed in Example 1.34. We propose a symmetric monoidal structure on $\mathbb{P}(S)$ with monoidal unit S and monoidal product given by intersection $A \cap B$.

Does it satisfy the conditions of Definition 2.1? \diamond

Exercise 2.18. Let $\text{Prop}^{\mathbb{N}}$ denote the set of all mathematical statements one can make about natural numbers. For example “ n is prime” is an element of $\text{Prop}^{\mathbb{N}}$, as is “ $n=2$ ” and “ $n \geq 11$ ”. Given $P, Q \in \text{Prop}^{\mathbb{N}}$, we say $P \leq Q$ if for all $n \in \mathbb{N}$, whenever $P(n)$ is true, so is $Q(n)$.

Define a monoidal unit and a monoidal product on $\text{Prop}^{\mathbb{N}}$ that satisfy the conditions of Definition 2.1. \diamond

The monoidal poset Cost As we said above, when we enrich in monoidal posets we see them as different ways to structure “getting from here to there”. We will explain this in more detail in Section 2.3. The following monoidal poset will eventually structure a notion of distance or cost for getting from here to there.

Example 2.19 (Lawvere’s monoidal poset). Let $[0, \infty]$ denote the set of nonnegative real numbers—such as 0, 1, 15.333, and 2π —together with ∞ . Consider the poset $([0, \infty], \geq)$, with the usual notion of \geq , where of course $\infty \geq x$ for all $x \in [0, \infty]$.

There is a monoidal structure on this poset, where the monoidal unit is 0 and the monoidal product is $+$. In particular, $x + \infty = \infty$ for any $x \in [0, \infty]$. Let’s call this monoidal poset

$$\mathbf{Cost} := ([0, \infty], \geq, 0, +)$$

because we can think of the elements of $[0, \infty]$ as costs. In terms of structuring “getting from here to there”, Lawvere’s monoidal poset seems to say “getting from a to b is a question of cost”. The monoidal unit being 0 will translate into saying that you can always get from a to a at no cost. The monoidal product being $+$ will translate into saying that the cost of getting from a to c is at most the cost of getting from a to b plus the cost of getting from b to c . Finally, the “at most” in the previous sentence is coming from the \geq . \diamond

The opposite of a monoidal poset One can take the opposite of any poset, just flip the order: $(X, \leq)^{\text{op}} := (X, \geq)$; see Example 1.42. Proposition 2.20 says that if the poset had a symmetric monoidal structure, so does its opposite.

Proposition 2.20. *Suppose $\mathcal{X} = (X, \leq)$ is a poset and $\mathcal{X}^{\text{op}} = (X, \geq)$ is its opposite. If (X, \leq, I, \otimes) is a symmetric monoidal poset then so is its opposite, (X, \geq, I, \otimes) .*

Proof. Suppose $x_1 \geq y_1$ and $x_2 \geq y_2$ in \mathcal{X}^{op} ; we need to show that $x_1 \otimes x_2 \geq y_1 \otimes y_2$. But by definition of opposite order, we have $y_1 \leq x_1$ and $y_2 \leq x_2$ in \mathcal{X} , and thus $y_1 \otimes y_2 \leq x_1 \otimes x_2$ in \mathcal{X} . Thus indeed $x_1 \otimes x_2 \geq y_1 \otimes y_2$ in \mathcal{X}^{op} . The other three conditions are even easier; see Exercise 2.21. \square

Exercise 2.21. Complete the proof of Proposition 2.20 by proving that the three remaining conditions of Definition 2.1 are satisfied. \diamond

Exercise 2.22. Since **Cost** is a symmetric monoidal poset, Proposition 2.20 says that **Cost**^{op} is too.

1. What is **Cost**^{op} as a poset?
2. What is its monoidal unit?
3. What is its monoidal product? \diamond

2.2.5 Monoidal monotone maps

Recall from Example 1.33 that for any poset (X, \leq) , there is an induced equivalence relation \cong on X , where $x \cong x'$ iff both $x \leq x'$ and $x' \leq x$.

Definition 2.23. Let $\mathcal{P} = (P, \leq_P, I_P, \otimes_P)$ and $\mathcal{Q} = (Q, \leq_Q, I_Q, \otimes_Q)$ be monoidal posets. A *monoidal monotone* from \mathcal{P} to \mathcal{Q} is a monotone map $f: (P, \leq_P) \rightarrow (Q, \leq_Q)$, satisfying two conditions:

- a. $I_Q \leq f(I_P)$, and
- b. $f(p_1) \otimes_Q f(p_2) \leq_Q f(p_1 \otimes_P p_2)$

for all $p_1, p_2 \in P$.

There are strengthenings of these conditions that are also important. If f satisfies the following conditions, it is called a *strong monoidal monotone*:

- a'. $I_Q \cong f(I_P)$, and
- b'. $f(p_1) \otimes_Q f(p_2) \cong f(p_1 \otimes_P p_2)$;

and if it satisfies the following conditions it is called a *strict monoidal monotone*:

- a''. $I_Q = f(I_P)$, and
- b''. $f(p_1) \otimes_Q f(p_2) = f(p_1 \otimes_P p_2)$.

Example 2.24. There is a monoidal monotone $i: (\mathbb{N}, \leq, 0, +) \rightarrow (\mathbb{R}, \leq, 0, +)$, where $i(n) = n$ for all $n \in \mathbb{N}$. It is clearly monotonic, $m \leq n$ implies $i(m) \leq i(n)$. It is even strict monoidal because $i(0) = 0$ and $i(m + n) = i(m) + i(n)$.

There is also a monoidal monotone $f: (\mathbb{R}, \leq, 0, +) \rightarrow (\mathbb{N}, \leq, 0, +)$ going the other way. Here $f(x) := \lfloor x \rfloor$ is the floor function, e.g. $f(3.14) = 3$. It is monotonic because $x \leq y$ implies $f(x) \leq f(y)$. Also $f(0) = 0$ and $f(x) + f(y) \leq f(x + y)$, so it is a monoidal monotone. But it is not strict or even strong because $f(0.5) + f(0.5) \leq f(0.5 + 0.5)$. \blacklozenge

Recall **Bool** = $(\mathbb{B}, \leq, \text{true}, \wedge)$ from Example 2.10 and **Cost** from Example 2.19. There is a monoidal monotone $g: \mathbf{Bool} \rightarrow \mathbf{Cost}$, given by $g(\text{false}) := \infty$ and $g(\text{true}) := 0$.

Exercise 2.25. 1. Check that the map $g: (\mathbb{B}, \leq, \text{true}, \wedge) \rightarrow ([0, \infty], \geq, 0, +)$ presented above indeed

- is monotonic,
- satisfies the condition a. of Definition 2.23, and
- satisfies the condition b. of Definition 2.23.

2. Is g strict? \diamond

Exercise 2.26. Let **Bool** and **Cost** be as above, and consider the following quasi-inverse functions $d, u: [0, \infty] \rightarrow \mathbb{B}$ defined as follows:

$$d(x) := \begin{cases} \text{false} & \text{if } x > 0 \\ \text{true} & \text{if } x = 0 \end{cases} \quad u(x) := \begin{cases} \text{false} & \text{if } x = \infty \\ \text{true} & \text{if } x < \infty \end{cases}$$

1. Is d monotonic?
2. Does d satisfy conditions a. and b. of Definition 2.23?
3. Is d strict?
4. Is u monotonic?
5. Does u satisfy conditions a. and b. of Definition 2.23?
6. Is u strict? ◇

Exercise 2.27. 1. Is $(\mathbb{N}, \leq, 1, *)$ a monoidal poset, where $*$ is the usual multiplication of natural numbers?
 2. If not, why not? If so, find a monoidal monotone $(\mathbb{N}, \leq, 0, +) \rightarrow (\mathbb{N}, \leq, 1, *)$. ◇

2.3 Enrichment

In this section we will introduce \mathcal{V} -categories, where \mathcal{V} is a symmetric monoidal poset. We will see that **Bool**-categories are posets, and that **Cost**-categories are a nice generalization of the notion of metric space.

2.3.1 \mathcal{V} -categories

Definition 2.28. Let $\mathcal{V} = (V, \leq, I, \otimes)$ be a symmetric monoidal poset. A \mathcal{V} -category \mathcal{X} consists of two constituents, satisfying two properties. To specify \mathcal{X} ,

- (i) one specifies a set $\text{Ob}(\mathcal{X})$, elements of which are called *objects*;
- (ii) for every two objects x, y , one specifies an element $\mathcal{X}(x, y) \in \mathcal{V}$, called the *hom-object*.²

The above constituents are required to satisfy two properties:

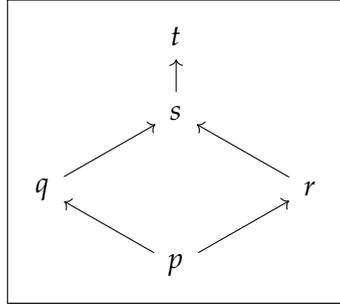
- (a) for every object $x \in \text{Ob}(\mathcal{X})$ we have $I \leq \mathcal{X}(x, x)$, and
- (b) for every three objects $x, y, z \in \text{Ob}(\mathcal{X})$, we have $\mathcal{X}(x, y) \otimes \mathcal{X}(y, z) \leq \mathcal{X}(x, z)$.

We call \mathcal{V} the *base of the enrichment* for \mathcal{X} or that \mathcal{X} is *enriched* in \mathcal{V} .

Example 2.29. As we shall see in the next subsection, from every poset we can construct a **Bool**-category, and vice versa. So, to get a feel for \mathcal{V} -categories, let us consider the

²The word “hom” is short for *homomorphism* and reflects the origins of this subject. A more descriptive name for $\mathcal{X}(x, y)$ might be *mapping object*, but we use “hom” mainly because it is an important word to know in the field.

poset generated by the Hasse diagram:



(2.16)

How does this correspond to a **Bool**-category \mathcal{X} ? Well, the objects of \mathcal{X} are simply the elements of the poset, and hence the set $\{p, q, r, s, t\}$. Next, for every pair of objects (x, y) we need an element of $\mathbb{B} = \{\text{false}, \text{true}\}$: simply take **true** if $x \leq y$, and **false** if otherwise. So for example, since $s \leq t$ and $t \not\leq s$, we have $\mathcal{X}(s, t) = \text{true}$ and $\mathcal{X}(t, s) = \text{false}$. Recalling that the monoidal unit I of **Bool** is **false**, it's straightforward to check that this obeys both (a) and (b), so we have a **Bool**-category.

In general, it's convenient to represent a \mathcal{V} -category \mathcal{X} with a square matrix. The rows and columns of the matrix correspond to the objects of \mathcal{X} , and the (x, y) th entry is simply the hom-object $\mathcal{X}(x, y)$. So, for example, the above poset shown left in Eq. (2.16) can be represented by the matrix

$i \setminus j$	p	q	r	s	t
p	true	true	true	true	true
q	false	true	false	true	true
r	false	false	true	true	true
s	false	false	false	true	true
t	false	false	false	false	true

2.3.2 Posets as Bool-categories

Our colleague Peter Gates has called category theory “a primordial ooze”, because so much of it can be defined in terms of other parts of it. There is nowhere to rightly call the beginning, because that beginning can be defined in terms of something else. So be it; this is part of the fun.

Theorem 2.30. *There is a one-to-one correspondence between posets and **Bool**-categories.*

Here we find ourselves in the ooze, because we are saying that posets are the same as **Bool**-categories, whereas **Bool** is itself a poset. “So then **Bool** is like... enriched in itself?” Yes, every poset, including **Bool**, is enriched in **Bool**, as we will now see.

Proof of Theorem 2.30. Let's check that we can construct a poset from any **Bool**-category. Since $\mathbb{B} = \{\text{false}, \text{true}\}$, Definition 2.28 says a **Bool**-category consists of two things:

1. a set $\text{Ob}(\mathcal{X})$, and

2. for every $x, y \in \text{Ob}(\mathcal{X})$ an element $\mathcal{X}(x, y) \in \mathbb{B}$, i.e. either $\mathcal{X}(x, y) = \text{true}$ or $\mathcal{X}(x, y) = \text{false}$.

We will use these two things to begin forming a poset whose set of elements are the objects of \mathcal{X} . So let's call the poset (X, \leq) , and let $X := \text{Ob}(\mathcal{X})$. For the \leq relation, let's say $x \leq y$ iff $\mathcal{X}(x, y) = \text{true}$. We have the makings of a poset, but for it to work, it must be reflexive and transitive. Let's see if we get these from the properties guaranteed by Definition 2.28:

- a. for every element $x \in X$ we have $\text{true} \leq \mathcal{X}(x, x)$,
- b. for every three elements $x, y, z \in X$ we have $\mathcal{X}(x, y) \wedge \mathcal{X}(y, z) \leq \mathcal{X}(x, z)$.

For $b \in \mathbf{Bool}$, if $\text{true} \leq b$ then $b = \text{true}$, so the first statement says $\mathcal{X}(x, x) = \text{true}$, which means $x \leq x$. For the second statement, one can consult Eq. (2.15). Since $\text{false} \leq b$ for all $b \in \mathbb{B}$, the only way statement b. has any force is if $\mathcal{X}(x, y) = \text{true}$ and $\mathcal{X}(y, z) = \text{true}$, in which case it forces $\mathcal{X}(x, z) = \text{true}$. This condition exactly translates as saying that $x \leq y$ and $y \leq z$ implies $x \leq z$. Thus we have obtained reflexivity and transitivity from the two axioms of **Bool**-categories.

In Example 2.29, we constructed a **Bool**-category from a poset. We leave it to the reader to generalise this example and show that the two constructions are inverses; see Exercise 2.31. \square

Exercise 2.31. Start with a poset (P, \leq) , and use it to define a **Bool**-category as we did in Example 2.29. In the proof of Theorem 2.30 we showed how to turn that **Bool**-category back into a poset. Show that doing so, you get the poset you started with. \diamond

2.3.3 Lawvere metric spaces

Metric spaces offer a precise way to describe spaces of points, each pair of which is separated by some distance. Here is the usual definition:

Definition 2.32. A *metric space* consists of:

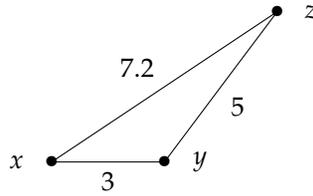
1. a set X , elements of which are called *points*, and
2. a function $d: X \times X \rightarrow \mathbb{R}_{\geq 0}$, where $d(x, y)$ is called the *distance between x and y* .

These constituents must satisfy four properties:

- (a) for every $x \in X$, we have $d(x, x) = 0$,
- (b) for every $x, y \in X$, if $d(x, y) = 0$ then $x = y$,
- (c) for every $x, y \in X$, we have $d(x, y) = d(y, x)$, and
- (d) for every $x, y, z \in X$, we have $d(x, y) + d(y, z) \geq d(x, z)$.

The fourth property is called the *triangle inequality*.

The triangle inequality says that when plotting a route from x to z , the distance is always at most what you get by choosing an intermediate point y and going $x \rightarrow y \rightarrow z$.



It can be invoked three different ways in the above picture: $3 + 5 \geq 7.2$, but also $5 + 7.2 \geq 3$ and $3 + 7.2 \geq 5$.

The triangle inequality wonderfully captures something about distance, as does the fact that $d(x, x) = 0$. However, the other conditions are not quite as general as we would like. Indeed, there are many examples of things that “should” be metric spaces, but which do not satisfy conditions b. or c. of Definition 2.32.

For example, what if we take X to be your neighborhood, but instead of measuring distance, you want $d(x, y)$ to measure *effort* to get from x to y . Then if there are any hills, the symmetry axiom, $d(x, y) \stackrel{?}{=} d(y, x)$, fails: it’s easier to get from x downhill to y then to go from y uphill to x .

Another way to find a model that breaks the symmetry axiom is to imagine that the elements of X are not points, but whole regions such as the US, Spain, and Boston. Say that the distance from region A to region B is understood using the setup “I will put you in an arbitrary part of A and you just have to get anywhere in B ; what is the distance in the worst-case scenario?” So $d(\text{US}, \text{Spain})$ is the distance from somewhere in the western US to the western tip of Spain: you just have to get into Spain, but you start in the worst possible part of the US for doing so.

Exercise 2.33. Which distance is bigger under the above description, $d(\text{Spain}, \text{US})$ or $d(\text{US}, \text{Spain})$? \diamond

This notion of distance, which is strongly related to something called *Hausdorff distance*,³ will again satisfy the triangle inequality, but it violates the symmetry condition. It also violates another condition, because $d(\text{Boston}, \text{US}) = 0$. No matter where you are in Boston, the distance to the nearest point of the US is 0. On the other hand, $d(\text{US}, \text{Boston}) \neq 0$.

Finally, one can imagine a use for distances that are not finite. In terms of my effort, the distance from here to Pluto is ∞ , and it would not be any better if Pluto was

³ The Hausdorff distance gives a metric on the set of all subsets $U \subseteq X$ of a given metric space (X, d) . One first defines

$$d_L(U, V) := \sup_{u \in U} \inf_{v \in V} d(u, v),$$

and this is exactly the formula we intend above; the result will be a Lawvere metric space. However, if one wants the Hausdorff distance to define a (symmetric) metric, as in Definition 2.32, one must take the above formula and symmetrize it: $d(U, V) := \max(d_L(U, V), d_L(V, U))$. We happen to see the unsymmetrized notion as more interesting.

still a planet. Similarly, in terms of Hausdorff distance, discussed above, the distance between two regions is often infinite, e.g. the distance between $\{r \in \mathbb{R} \mid r < 0\}$ and $\{0\}$ as subsets of (\mathbb{R}, d) is infinite.

When we drop conditions (b) and (c) and allow for infinite distances, we get the following relaxed notion of metric space, first proposed by Lawvere. Recall the symmetric monoidal poset $\mathbf{Cost} = ([0, \infty], \geq, 0, +)$ from Example 2.19.

Definition 2.34. A *Lawvere metric space* is a \mathbf{Cost} -category.

This is a very compact definition, but it packs a punch. Let's work out what it means, by relating it to the usual definition of metric space. By Definition 2.28, a \mathbf{Cost} -category \mathcal{X} consists of:

- (i) a set $\text{Ob}(\mathcal{X})$,
- (ii) for every $x, y \in \text{Ob}(\mathcal{X})$ an element $\mathcal{X}(x, y) \in [0, \infty]$.

Here the set $\text{Ob}(\mathcal{X})$ is playing the role of the set of points, and $\mathcal{X}(x, y) \in [0, \infty]$ is playing the role of distance, so let's write

$$X := \text{Ob}(\mathcal{X}) \quad d(x, y) := \mathcal{X}(x, y).$$

The properties of a category enriched in \mathbf{Cost} are:

- (a) $0 \geq d(x, x)$ for all $x \in X$, and
- (b) $d(x, y) + d(y, z) \geq d(x, z)$ for all $x, y, z \in X$.

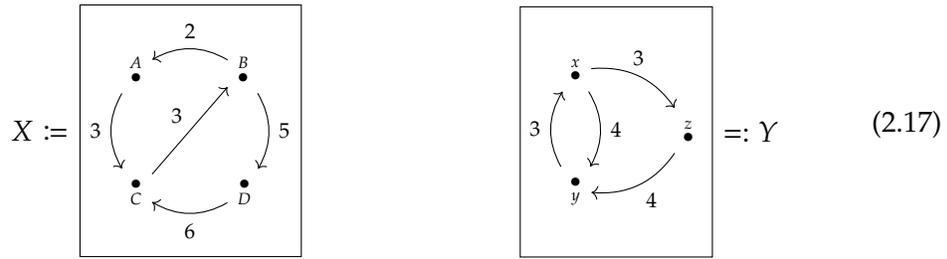
Since $d(x, x) \in [0, \infty]$, if $0 \geq d(x, x)$ then $d(x, x) = 0$. So the first condition is equivalent to the first condition from Definition 2.32, namely $d(x, x) = 0$. The second condition is the triangle inequality.

Example 2.35. The set \mathbb{R} of real numbers can be given a metric space structure, and hence a Lawvere metric space structure. Namely $d(x, y) := |y - x|$, the absolute value of the difference. So $d(3, 7) = 4$. \blacklozenge

Exercise 2.36. Consider the symmetric monoidal poset $(\mathbb{R}_{\geq 0}, \geq, 0, +)$, which is almost the same as \mathbf{Cost} , except it does not include ∞ . How would you characterize the difference between a Lawvere metric space and a category enriched in $(\mathbb{R}_{\geq 0}, \geq, 0, +)$? \blacklozenge

Presenting metric spaces with weighted graphs One can convert any weighted graph—a graph whose edges are labeled with numbers $w \geq 0$ —into a Lawvere metric space. In fact, we shall consider these as graphs labelled with elements of the $[0, \infty]$, and more precisely call them \mathbf{Cost} -weighted graphs. One might think of such a graph as describing a city with some one-way roads (a two-way road is modeled as two one-way roads), each having some length. For example, consider the following weighted

graphs:



Given a weighted graph, one forms a metric d_X on its set X of vertices by setting $d(p, q)$ to be the length of the shortest path from p to q . For example, here is the table of distances for Y

$d(\nearrow)$	x	y	z
x	0	4	3
y	3	0	6
z	7	4	0

(2.18)

Exercise 2.37. Fill out the following table of distances in the weighted graph X from Eq. (2.17)

$d(\nearrow)$	A	B	C	D
A	0	?	?	?
B	2	?	5	?
C	?	?	?	?
D	?	?	?	?

◇

Above we converted a weighted graph G , e.g. as shown in Eq. (2.17), into a table of distances, but this takes a bit of thinking. There is a more direct construction for getting a square matrix M_G out of G , whose rows and columns are indexed by the vertices of G . To do so, set M_G to be 0 along the diagonal, to be ∞ wherever an edge is missing, and to be the edge weight if there is an edge.

For example, the matrix associated to Y in Eq. (2.17) would be

\nearrow	x	y	z
x	0	4	3
y	3	0	∞
z	∞	4	0

(2.19)

As soon as you see how we did this, you'll understand that it takes no thinking to turn a weighted graph G into a matrix M_G in this way. We will see later in Section 2.5.3 that the more difficult "distance matrices" d_Y , such as Eq. (2.18), can be obtained from the easy graph matrices M_Y , such as Eq. (2.19), by repeated a certain sort of "matrix multiplication".

Exercise 2.38. Fill out the matrix M_X associated to the graph X in Eq. (2.17):

$$M_X = \begin{array}{c|cccc} \nearrow & A & B & C & D \\ \hline A & 0 & ? & ? & ? \\ B & 2 & 0 & \infty & ? \\ C & ? & ? & ? & ? \\ D & ? & ? & ? & ? \end{array} \quad \diamond$$

2.3.4 \mathcal{V} -variations on posets and metric spaces

We have told the story of **Bool**. But in Section 2.2.4 we gave examples of many other monoidal posets, and each one serves as the base for a kind of enriched category. Which of them are useful? Something only becomes useful when someone finds a use for it. We will find uses for some and not others, though we encourage readers to think about what it would mean to enrich in the various monoidal categories discussed above; maybe they can find a use we have not explored.

Exercise 2.39. Recall the monoidal poset $\mathbf{NMY} := (P, \leq, \text{yes}, \min)$ from Exercise 2.16. Interpret what a category enriched in \mathbf{NMY} would be. \diamond

Exercise 2.40. Let M be a set and let $\mathcal{M} := (\mathbb{P}(M), \subseteq, M, \cap)$ be the monoidal poset whose elements are subsets of M .

Someone gives the following interpretation, “for any set M , imagine it as the set of modes of transportation (e.g. car, boat, foot). Then a category \mathcal{X} enriched in \mathcal{M} tells you all the modes that will get you from a all the way to b , for any two points $a, b \in \text{Ob}(\mathcal{X})$.”

1. Draw a graph with four vertices and four or five edges, each labeled with a subset of $M = \{\text{car}, \text{boat}, \text{foot}\}$.
2. This corresponds to a \mathcal{M} -category; call it \mathcal{X} . Write out the corresponding four-by-four matrix of hom-objects.
3. Does the person’s interpretation look right, or is it subtly mistaken somehow? \diamond

Exercise 2.41. Consider the poset $W := (\mathbb{N} \cup \{\infty\}, \leq, \infty, \min)$.

1. Draw a small graph labeled by elements of W and compute the corresponding distance table. This will give you a feel for how W works.
2. Make up a interpretation, like that in Exercise 2.40, for how to imagine enrichment in W . \diamond

2.4 Constructions on enriched categories

Now that we have a good intuition for what \mathcal{V} -categories are, we give three examples of what can be done with \mathcal{V} -categories. The first (Section 2.4.1) is known as change of base. This allows us to use a monoidal monotone $f: \mathcal{V} \rightarrow \mathcal{W}$ to construct \mathcal{W} -categories from \mathcal{V} -categories. The second construction (Section 2.4.2), that of \mathcal{V} -functors, allows us to complete the analogy: a poset is to a **Bool**-category as a monotone map is to

what? The third construction (Section 2.4.2) is known as a \mathcal{V} -product, and gives us a way of combining two \mathcal{V} -categories.

2.4.1 Changing the base of enrichment

Any monoidal monotone $\mathcal{V} \rightarrow \mathcal{W}$ between monoidal posets lets us convert \mathcal{V} -categories into \mathcal{W} -categories.

Construction 2.42. Let $f: \mathcal{V} \rightarrow \mathcal{W}$ be a monoidal monotone. Given a \mathcal{V} -category C , one forms the associated \mathcal{W} -category, say C_f as follows.

1. we take the same objects: $\text{Ob}(C_f) := \text{Ob}(C)$,
2. for any $c, d \in \text{Ob}(C)$, put $C_f(c, d) := f(C(c, d))$

This construction C_f does indeed obey the definition of a \mathcal{W} -category, as can be seen by applying Definitions 2.23 and 2.28:

- a. $I_W \leq f(I_V) \leq f(C(c, c)) = C_W(c, c)$ for every $c \in C$, and
- b. for every $c, d, e \in \text{Ob}(C)$ we have

$$\begin{aligned}
 C_f(c, d) \otimes_W C_f(d, e) &= f(C(c, d)) \otimes_W f(C(d, e)) && \text{def. of } C_f \\
 &\leq f(C(c, d) \otimes_V C(d, e)) && \text{Definition 2.23} \\
 &\leq f(C(c, e)) && \text{Definition 2.28} \\
 &= C_f(c, e) && \text{def. of } C_f
 \end{aligned}$$

Example 2.43. As an example, consider the function $f: [0, \infty] \rightarrow \{\text{false}, \text{true}\}$ given by

$$f(x) := \begin{cases} \text{true} & \text{if } x = 0 \\ \text{false} & \text{if } x > 0 \end{cases} \quad (2.20)$$

It is easy to check that f is monotonic and that f preserves the monoidal product and monoidal unit; that is, it's easy to show that f is a monoidal functor. Thus f lets us convert Lawvere metric spaces into posets. \blacklozenge

Exercise 2.44. Recall the “regions of the world” Lawvere metric space from Exercise 2.33 and the text above it. We just learned that we can convert it to a poset. Draw the Hasse diagram for the poset corresponding to the regions: US, Spain, and Boston. \blacklozenge

Exercise 2.45. 1. Find another monoidal functor $g: \mathbf{Cost} \rightarrow \mathbf{Bool}$ different from the one defined in Eq. (2.20).

2. Using Construction 2.42, both your monoidal functor g and the functor f in Eq. (2.20) can be used to convert a Lawvere metric space into a poset. Find a Lawvere metric space X on which they give different answers, $X_f \neq X_g$. \blacklozenge

2.4.2 Enriched functors

A functor is the most important type of relationship between categories.

Definition 2.46. Let \mathcal{X} and \mathcal{Y} be \mathcal{V} -categories. A \mathcal{V} -functor from \mathcal{X} to \mathcal{Y} , denoted $F: \mathcal{X} \rightarrow \mathcal{Y}$, consists of one constituent:

- (i) a function $F: \text{Ob}(\mathcal{X}) \rightarrow \text{Ob}(\mathcal{Y})$

subject to one constraint

- (a) for all $x_1, x_2 \in \text{Ob}(\mathcal{X})$, one has $\mathcal{X}(x_1, x_2) \leq \mathcal{Y}(F(x_1), F(x_2))$.

Example 2.47. For example, we have said several times—e.g. in Theorem 2.30—that posets are **Bool**-categories, where $\mathcal{X}(x_1, x_2) = \text{true}$ is denoted $x_1 \leq x_2$. One would hope that monotone maps between posets would correspond exactly to **Bool**-functors, and that’s true. A monotone map $(X, \leq_X) \rightarrow (Y, \leq_Y)$ is a function $F: X \rightarrow Y$ such that for every $x_1, x_2 \in X$, if $x_1 \leq x_2$ then $F(x_1) \leq F(x_2)$. In other words $\mathcal{X}(x_1, x_2) \leq \mathcal{Y}(F(x_1), F(x_2))$, which matches Definition 2.46 perfectly. \blacklozenge

Remark 2.48. In fact, we have what is called an *equivalence* of categories between the category of posets and the category of **Bool**-categories. In the next chapter we will develop the ideas to state what this means precisely (Remark 2.48).

Example 2.49. Lawvere metric spaces are **Cost**-categories. The definition of **Cost**-functor should hopefully return a nice notion—a “friend”—from the theory of metric spaces, and it does: it recovers the notion of Lipschitz function. A Lipschitz function is one where the distance between any pair of points does not increase. That is, given Lawvere metric spaces (X, d_X) and (Y, d_Y) , a **Cost**-functor between them is a function $F: X \rightarrow Y$ such that for every $x_1, x_2 \in X$ we have $d_X(x_1, x_2) \geq d_Y(F(x_1), F(x_2))$. \blacklozenge

Exercise 2.50. Just like we can talk of opposite posets, we can talk of the opposite of any \mathcal{V} -category. The *opposite* of a \mathcal{V} -category \mathcal{X} is denoted \mathcal{X}^{op} and is defined by

- (i.) $\text{Ob}(\mathcal{X}^{\text{op}}) := \text{Ob}(\mathcal{X})$, and
(ii.) for all $x, y \in \mathcal{X}$, we have $\mathcal{X}^{\text{op}}(x, y) := \mathcal{X}(y, x)$.

Similarly, we can talk of daggers. A *dagger structure* on a \mathcal{V} -category \mathcal{X} is a functor $\dagger: \mathcal{X} \rightarrow \mathcal{X}^{\text{op}}$ that is identity on objects and such that $\dagger \circ \dagger = \text{id}_{\mathcal{X}}$.

Recall that an ordinary metric space (X, d) is a Lawvere metric space with some extra properties; see Definition 2.32. One of these properties is symmetry: $d(x, y) = d(y, x)$ for every $x, y \in X$. What if we have a Lawvere metric space (X, d) such that the identity function $\text{id}_X: X \rightarrow X$ is a **Cost**-functor $(X, d) \rightarrow (X, d)^{\text{op}}$. Is this exactly the same as the symmetry property?

1. Show that a skeletal dagger **Cost**-category is a metric space.
2. Make sense of the following analogy: “posets are to sets as Lawvere metric spaces are to metric spaces.” \blacklozenge

2.4.3 Product \mathcal{V} -categories

If \mathcal{V} is a symmetric monoidal poset and \mathcal{X} and \mathcal{Y} are \mathcal{V} -categories, then we can define their \mathcal{V} -product, which is a new \mathcal{V} -category.

Definition 2.51. Let \mathcal{X} and \mathcal{Y} be \mathcal{V} -categories. Define their \mathcal{V} -product, or simply product, to be the \mathcal{V} -category $\mathcal{X} \times \mathcal{Y}$ with

1. $\text{Ob}(\mathcal{X} \times \mathcal{Y}) := \text{Ob}(\mathcal{X}) \times \text{Ob}(\mathcal{Y})$,
2. $(\mathcal{X} \times \mathcal{Y})((x, y), (x', y')) := \mathcal{X}(x, x') \otimes \mathcal{Y}(y, y')$,

for two objects (x, y) and (x', y') in $\text{Ob}(\mathcal{X} \times \mathcal{Y})$.

Product \mathcal{V} -categories are indeed \mathcal{V} -categories (Definition 2.28); see Exercise 2.52.

Exercise 2.52. Let $\mathcal{X} \times \mathcal{Y}$ be the \mathcal{V} -product as in Definition 2.51.

1. Check that for every object $(x, y) \in \text{Ob}(\mathcal{X} \times \mathcal{Y})$ we have $I \leq (\mathcal{X} \times \mathcal{Y})((x, y), (x, y))$.
2. Check that for every three objects (x_1, y_1) , (x_2, y_2) , and (x_3, y_3) , we have $(\mathcal{X} \times \mathcal{Y})((x_1, y_1), (x_2, y_2)) \otimes (\mathcal{X} \times \mathcal{Y})((x_2, y_2), (x_3, y_3)) \leq (\mathcal{X} \times \mathcal{Y})((x_1, y_1), (x_3, y_3))$. \diamond

Thus when taking the product of two posets $(P, \leq_P) \times (Q, \leq_Q)$, we say that $(p_1, q_1) \leq (p_2, q_2)$ iff both $p_1 \leq p_2$ AND $q_1 \leq q_2$; the AND is the monoidal product \otimes from of **Bool**. Thus the product of posets is an example of a **Bool**-product.

Example 2.53. Let \mathcal{X} and \mathcal{Y} be the Lawvere metric spaces defined by the following weighted graphs:

$$\mathcal{X} := \boxed{\begin{array}{ccc} A & \xrightarrow{2} & B & \xrightarrow{3} & C \\ \bullet & & \bullet & & \bullet \end{array}} \quad \boxed{\begin{array}{c} p \\ \bullet \\ \left. \begin{array}{c} \curvearrowright \\ \downarrow \\ \curvearrowleft \end{array} \right\} \begin{array}{l} 5 \\ 8 \end{array} \\ \bullet \\ q \end{array}} =: \mathcal{Y} \tag{2.21}$$

Their product is defined by taking the product of their sets of objects, so there are six objects in $\mathcal{X} \times \mathcal{Y}$. And the distance $d_{\mathcal{X} \times \mathcal{Y}}((x, y), (x', y'))$ between any two points is given by the sum $d_{\mathcal{X}}(x, x') + d_{\mathcal{Y}}(y, y')$.

Examine the following graph, and make sure you understand how easy it is to derive from the weighted graphs for \mathcal{X} and \mathcal{Y} in Eq. (2.21):

$$\mathcal{X} \times \mathcal{Y} = \boxed{\begin{array}{ccccc} (A, p) & \xrightarrow{2} & (B, p) & \xrightarrow{3} & (C, p) \\ \bullet & & \bullet & & \bullet \\ \left. \begin{array}{c} \curvearrowright \\ \downarrow \\ \curvearrowleft \end{array} \right\} \begin{array}{l} 5 \\ 8 \end{array} & & \left. \begin{array}{c} \curvearrowright \\ \downarrow \\ \curvearrowleft \end{array} \right\} \begin{array}{l} 5 \\ 8 \end{array} & & \left. \begin{array}{c} \curvearrowright \\ \downarrow \\ \curvearrowleft \end{array} \right\} \begin{array}{l} 5 \\ 8 \end{array} \\ (A, q) & \xrightarrow{2} & (B, q) & \xrightarrow{3} & (C, q) \\ \bullet & & \bullet & & \bullet \end{array}} \quad \blacklozenge$$

Exercise 2.54. Consider \mathbb{R} as a Lawvere metric space, i.e. as a **Cost**-category (see Example 2.35). Form the **Cost**-product $\mathbb{R} \times \mathbb{R}$. What is the distance from $(5, 6)$ to $(-1, 4)$? Hint: apply Definition 2.51; the answer is not $\sqrt{40}$. \diamond

In terms of matrices, \mathcal{V} -products are also quite straightforward. They generalize what is known as the Kronecker product of matrices. The matrices for \mathcal{X} and \mathcal{Y} in

Eq. (2.21) are shown below

$$\begin{array}{c|ccc} \mathcal{X} & A & B & C \\ \hline A & 0 & 2 & 5 \\ B & \infty & 0 & 3 \\ C & \infty & \infty & 0 \end{array} \qquad \begin{array}{c|cc} \mathcal{Y} & p & q \\ \hline p & 0 & 5 \\ q & 8 & 0 \end{array}$$

and their product is as follows:

$$\begin{array}{c|ccc|ccc} \mathcal{X} \times \mathcal{Y} & (A, p) & (B, p) & (C, p) & (A, q) & (B, q) & (C, q) \\ \hline (A, p) & 0 & 2 & 5 & 5 & 7 & 10 \\ (B, p) & \infty & 0 & 3 & \infty & 5 & 8 \\ (C, p) & \infty & \infty & 0 & \infty & \infty & 5 \\ \hline (A, q) & 8 & 10 & 13 & 0 & 2 & 5 \\ (B, q) & \infty & 8 & 11 & \infty & 0 & 3 \\ (C, q) & \infty & \infty & 8 & \infty & \infty & 0 \end{array}$$

We have drawn the product matrix as a block matrix, where there is one block—shaped like \mathcal{X} —for every entry of \mathcal{Y} . Make sure you can see each block as the \mathcal{X} -matrix shifted by an entry in \mathcal{Y} . This comes directly from the formula from Definition 2.51 and the fact that the \otimes operation in **Cost** is $+$.

2.5 Computing presented \mathcal{V} -categories with matrix multiplication

In Section 2.3.3 we promised a straightforward way to construct the matrix representation of a **Cost**-category from a **Cost**-weighted graph. To do this, we use a generalized matrix multiplication. We shall show that this works, not just for **Cost**, but also for **Bool**, and many other monoidal posets. The condition is that the poset must have the property of being a commutative quantale. These are posets with all joins, plus one additional ingredient, being *monoidal closed*, which we define next in Section 2.5.1. The definition of quantales will be given in Section 2.5.2.

2.5.1 Monoidal closed posets

The definition of enrichment in a symmetric monoidal poset \mathcal{V} makes sense, regardless of \mathcal{V} . But that does not mean that any base is as useful as any other. In this section we define closed monoidal categories, which in particular enrich themselves!

Definition 2.55. A symmetric monoidal poset $\mathcal{V} = (V, \leq, I, \otimes)$ is called *symmetric monoidal closed* (or just *closed*) if, for every two elements $v, w \in V$, there is an element $v \multimap w$ in \mathcal{V} , called the *hom-element*, with the property

$$(a \otimes v) \leq w \quad \text{iff} \quad a \leq (v \multimap w). \quad (2.22)$$

for all $a, v, w \in V$.

Remark 2.56. The term closed refers to the fact that a hom-element can be constructed; the idea is that the poset can be seen as closed under the ‘operation’ of ‘taking homs’. In later chapters we’ll meet the closely related concepts of compact closed categories (Definition 4.41) and cartesian closed categories (Section 7.2.1) that make this idea more precise.

Remark 2.57. Note that condition Eq. (2.22) says we have a Galois connection in the sense of Definition 1.70. In particular, it says that a monoidal poset is monoidal closed if, given any $v \in V$, the monotone map $- \otimes v: V \rightarrow V$ given by multiplying with v has a right adjoint. We write this right adjoint $v \multimap -: V \rightarrow V$.

Example 2.58. The monoidal poset **Cost** = $([0, \infty], \geq, 0, +)$ is monoidal closed. Indeed, for any $x, y \in [0, \infty]$, define $x \multimap y := \max(0, y - x)$. Then, for any $a, x, y \in [0, \infty]$, we have

$$a + x \geq y \quad \text{iff} \quad a \geq y - x \quad \text{iff} \quad \max(0, a) \geq \max(0, y - x) \quad \text{iff} \quad a \geq (x \multimap y)$$

so \multimap satisfies the condition of Eq. (2.22).

Note that we have not considered subtraction in **Cost** before; we can in fact use monoidal closure to *define* subtraction in terms of the poset order and monoidal structure! ♦

Exercise 2.59. Show that **Bool** = $(\mathbb{B}, \leq, \text{true}, \wedge)$ is monoidal closed. ♦

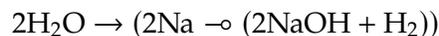
Example 2.60. A non-example is $(\mathbb{B}, \leq, \text{false}, \vee)$: it has all joins, as you showed in Exercise 2.59, but it is not monoidal closed. Indeed, suppose we had a \multimap operator as in Definition 2.55. Note that $\text{false} \leq p \multimap q$, for any p, q no matter what \multimap is, because false is less than everything.

Then using $a = \text{false}$, $p = \text{true}$, and $q = \text{false}$, we get a contradiction: $(a \vee p) \not\leq q$ and yet $a \leq (p \multimap q)$. ♦

Example 2.61. We started this chapter talking about resource theories. What does the closed structure look like in that case? For example, in chemistry it would say that for every two material collections c, d one can form a material collection $c \multimap d$ with the property that for any a , one has

$$a + c \rightarrow d \quad \text{if and only if} \quad a \rightarrow (c \multimap d).$$

Or more down to earth, since we have the reaction $2\text{H}_2\text{O} + 2\text{Na} \rightarrow 2\text{NaOH} + \text{H}_2$, we must also have



So from just two molecules of water, you can form a certain substance, but not many substances fit the bill. But it is not so far-fetched: this new substance $R = (2\text{Na} \multimap (2\text{NaOH} + \text{H}_2))$ is not really a substance, but the fact that a “sodium to sodium-hydroxide-plus-hydrogen” reaction is possible.

We leave it to the reader to think about what a category enriched in one of these resource theories is. We do not claim that this notion has much utility, but we also

don't claim that it doesn't; perhaps the innovative reader might find some use for it!

◆

Proposition 2.62. *Suppose $\mathcal{V} = (V, \leq, I, \otimes)$ is a symmetric monoidal poset, and that it is closed. Then*

1. *For every $v \in V$, the monotone map $- \otimes v: (V, \leq) \rightarrow (V, \leq)$ is left adjoint to $v \multimap -: (V, \leq) \rightarrow (V, \leq)$.*
2. *For any element $v \in V$ and set of elements $A \subseteq V$, we have*

$$\left(v \otimes \bigvee_{a \in A} a \right) \cong \bigvee_{a \in A} (v \otimes a). \quad (2.23)$$

3. *For any $v, w \in V$, we have $v \otimes (v \multimap w) \leq w$.*
4. *For any $v \in V$, we have $v \cong (I \multimap v)$.*
5. *For any $u, v, w \in V$, we have $(u \multimap v) \otimes (v \multimap w) \leq (u \multimap w)$.*

Proof. We go through the claims in order.

1. The definition of $(- \otimes v)$ being left adjoint to $(v \multimap -)$ is exactly the condition Eq. (2.22); see Definition 1.70.
2. This follows from 1, using Proposition 1.84.
3. By reflexivity, $(v \multimap w) \leq (v \multimap w)$, so $(v \multimap w) \otimes v \leq w$, and we are done by symmetry.
4. Since $v \otimes I = v \leq v$, condition Eq. (2.22) says $v \leq (I \multimap v)$. For the other direction, we have $(I \multimap v) = I \otimes (I \multimap v) \leq v$ by 3.
5. To obtain this inequality, we just need $u \otimes (u \multimap v) \otimes (v \multimap w) \leq w$. But this follows by two applications of 3.

□

Remark 2.63. We can consider \mathcal{V} to be enriched in itself. That is, for every $v, w \in \text{Ob}(\mathcal{V})$, we can define $\mathcal{V}(v, w) := (v \multimap w) \in \mathcal{V}$. For this to really be an enrichment, we just need to check the two conditions of Definition 2.28. The first condition $I \leq \mathcal{X}(x, x) = (x \multimap x)$ is satisfied because $I \otimes x \leq x$. The second condition is satisfied by Proposition 2.62 #5.

2.5.2 Commutative quantales

To perform matrix multiplication over a monoidal poset, we need one more thing: joins.

Definition 2.64. A *commutative quantale* is a symmetric monoidal closed poset that has all joins: $\bigvee A$ exists for every $A \subseteq V$. In particular, we denote the empty join by $0 := \bigvee \emptyset$.

Whenever we speak of quantales in this book, we mean commutative quantales. We will try to remind the reader of that. There are very interesting applications of noncommutative quantales; see Section 2.6.

Example 2.65. In Example 2.58, we saw that **Cost** is monoidal closed. To check whether **Cost** is a quantale, we take an arbitrary set of elements $A \subseteq [0, \infty]$ and ask if it has a join $\bigvee A$. To be a join, it needs to satisfy two properties:

- $a \geq \bigvee A$ for all $a \in A$, and
- if $b \in [0, \infty]$ is any element such that $a \geq b$ for all $a \in A$, then $\bigvee A \geq b$.

In fact we can define such a join: it is typically called the *infimum*, or greatest lower bound, of A . For example, if $A = \{2, 3\}$ then $\bigvee A = 2$. We have joins for infinite sets too: if $B = \{2.5, 2.05, 2.005, \dots\}$, its infimum is 2. Finally, in order to say that $([0, \infty], \geq)$ has all joins, we need a join to exist for the empty set $A = \emptyset$ too. The first condition becomes vacuous—there are no a 's in A —but the second condition says that for any $b \in B$ we have $\bigvee \emptyset \geq b$; this means $\bigvee \emptyset = \infty$.

Thus indeed $([0, \infty], \geq)$ has all joins, so **Cost** is a quantale. \blacklozenge

Remark 2.66. One can personify the notion of commutative quantale as a kind of navigator. A navigator is someone who understands “getting from one place to another”. Different navigators may care about or understand different aspects—whether one can get from A to B , how much time it will take, what modes of travel will work, etc.—but they certainly have some commonalities. Most importantly, a navigator needs to be able to read a map: given routes A to B and B to C , they understand how to get a route A to C . And they know how to search over the space of way-points to get from A to C . These will correspond to the monoidal product and the join operations, respectively.

Exercise 2.67. Show that **Bool** = $(\mathbb{B}, \leq, \text{true}, \wedge)$ is a quantale. \blacklozenge

Exercise 2.68. Let S be a set and recall the powerset monoidal poset $(\mathbb{P}(S), \subseteq, S, \cap)$ from Exercise 2.17. Is it a quantale? \blacklozenge

Example 2.69. Suppose \mathcal{V} is a commutative quantale that also has all meets.⁴ In this case, for any two \mathcal{V} -categories \mathcal{X} and \mathcal{Y} , there is a \mathcal{V} -category of \mathcal{V} -functors and certain constructions known as \mathcal{V} -natural transformations.

As an application, the notion of Hausdorff distance can be generalized when **Cost** is replaced by \mathcal{V} . If $U \subseteq X$ and $V \subseteq X$, the usual formula is shown on the left and its generalization is shown on the right:

$$d(U, V) := \sup_{u \in U} \inf_{v \in V} d(u, v) \qquad X(U, V) := \bigwedge_{u \in U} \bigvee_{v \in V} X(u, v).$$

For example, if $\mathcal{V} = \mathbf{Bool}$, the Hausdorff distance between sub-posets U and V answers the question “can I get into V from every $u \in U$ ”, i.e. $\forall u \in U. \exists v \in V. u \leq v$. Suppose we instead use $\mathcal{V} = \mathbb{P}(M)$ with its interpretation as modes of transportation, as in Exercise 2.40. Then the Hausdorff “distance” $d(U, V) \in \mathbb{P}(M)$ tells us those modes of transportation that will get us into V from every point in U . \blacklozenge

⁴We might call this a *cosmological quantale*, in keeping with the notion of *cosmos* from the category theory literature.

Proposition 2.70. *Suppose $\mathcal{V} = (V, \leq, I, \otimes)$ is any symmetric monoidal poset that has all joins. Then \mathcal{V} is closed—and hence a quantale—if and only if \otimes distributes over joins; i.e. if Eq. (2.23) holds for all $A \subseteq V$ and $v \in V$.*

Proof. We showed one direction in Proposition 2.62, #2: if \mathcal{V} is monoidal closed then Eq. (2.23) holds. We need to show that if the equation holds then $- \otimes v: V \rightarrow V$ has a right adjoint $v \multimap -$, but this is just the adjoint functor theorem, Proposition 1.88. It says we can define $v \multimap w$ to be

$$v \multimap x := \bigvee_{\{a \in V \mid a \otimes v \leq x\}} a. \quad \square$$

2.5.3 Matrix multiplication in a quantale

A quantale $\mathcal{V} = (V, \leq, I, \otimes)$, as defined in Definition 2.55, provides what is necessary to perform matrix multiplication.⁵ The usual formula for matrix multiplication is:

$$(M * N)(i, k) = \sum_j M(i, j) * N(j, k). \quad (2.24)$$

We will get a formula where joins stand in for the sum \sum , and \otimes stands in for the product operation $*$. Recall our convention of writing $0 := \bigvee \emptyset$.

Exercise 2.71. 1. What is $\bigvee \emptyset$, which we generally denote 0 , in the case

- a. $\mathcal{V} = \mathbf{Bool} = (\mathbb{B}, \leq, \mathbf{true}, \wedge)$?
 - b. $\mathcal{V} = \mathbf{Cost} = ([0, \infty], \geq, 0, +)$?
2. What is the join $x \vee y$ in the case
- a. $\mathcal{V} = \mathbf{Bool}$, and $x, y \in \mathbb{B}$ are booleans?
 - b. $\mathcal{V} = \mathbf{Cost}$, and $x, y \in [0, \infty]$ are distances. ◇

Definition 2.72. Let \mathcal{V} be a quantale. Given sets X and Y , a *matrix with entries in \mathcal{V}* , or simply a *\mathcal{V} -matrix*, is a function $M: X \times Y \rightarrow V$. For any $x \in X$ and $y \in Y$, we call $M(x, y)$ the *(x, y) -entry*.

Here is how you multiply \mathcal{V} -matrices. Suppose $M: X \times Y \rightarrow V$ and $N: Y \times Z \rightarrow V$ are V matrices. Their product is defined to be the matrix $(M * N): X \times Z \rightarrow V$, whose entries are given by the formula

$$(M * N)(x, z) := \bigvee_{y \in Y} M(x, y) \otimes N(y, z). \quad (2.25)$$

Note how similar this is to Eq. (2.24).

Example 2.73. Let $\mathcal{V} = \mathbf{Bool}$. Here is an example of matrix multiplication $M * N$, where $X = \{1, 2, 3\}$, $Y = \{1, 2\}$, and $Z = \{1, 2, 3\}$, and where $M: X \times Y \rightarrow \mathbb{B}$ and

⁵This works for noncommutative quantales as well.

$N: Y \times Z \rightarrow \mathbf{Bool}$:

$$\begin{pmatrix} \text{false} & \text{false} \\ \text{false} & \text{true} \\ \text{true} & \text{true} \end{pmatrix} * \begin{pmatrix} \text{true} & \text{true} & \text{false} \\ \text{true} & \text{false} & \text{true} \end{pmatrix} = \begin{pmatrix} \text{false} & \text{false} & \text{false} \\ \text{true} & \text{false} & \text{true} \\ \text{true} & \text{true} & \text{true} \end{pmatrix} \quad \blacklozenge$$

The identity V -matrix on a set X is $I_X: X \times X \rightarrow V$ given by

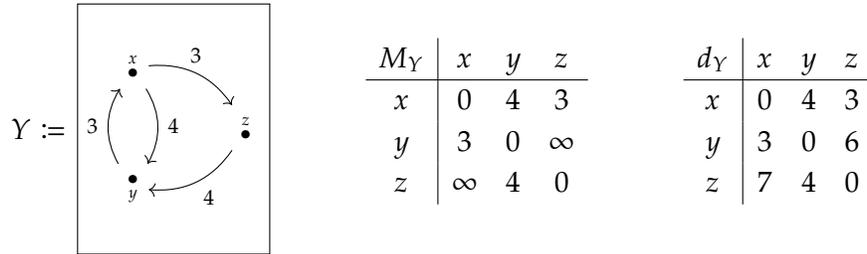
$$I_X(x, y) := \begin{cases} I & \text{if } x = y \\ 0 & \text{if } x \neq y. \end{cases}$$

Exercise 2.74. Let $\mathcal{V} = (V, \leq, I, \otimes)$ be a quantale. Use Eq. (2.25) and Proposition 2.62 to prove the following.

1. Show that for any sets X and Y and V -matrix $M: X \times Y \rightarrow V$, one has $I_X * M = M$.
2. Prove the associative law: for any matrices $M: W \times X \rightarrow V$, $N: X \times Y \rightarrow V$, and $P: Y \times Z \rightarrow V$, one has $(M * N) * P = M * (N * P)$.

\(\blacklozenge\)

Recall the weighted graph Y from Eq. (2.17). One can read off the associated matrix M_Y , and one can calculate the associated metric d_Y :



Here we explain how to obtain d_Y from M_Y .

The matrix M_Y can be thought of as recording paths consisting of either 0 or 1 edge-traversal: the diagonals being 0 mean we can get from x to x without traversing edges. When we can get from x to y in one edge we record its length in M_Y , otherwise we use ∞ .

When we multiply M_Y by itself using the formula Eq. (2.25), the result M_Y^2 tells us about paths with up to 2 edge-traversals. Similarly M_Y^3 tells us about paths with up to 3 edge-traversals:

$$M_Y^2 = \begin{array}{c|ccc} \nearrow & x & y & z \\ \hline x & 0 & 4 & 3 \\ y & 3 & 0 & 6 \\ z & 7 & 4 & 0 \end{array} \qquad M_Y^3 = \begin{array}{c|ccc} \nearrow & x & y & z \\ \hline x & 0 & 4 & 3 \\ y & 3 & 0 & 6 \\ z & 7 & 4 & 0 \end{array}$$

One sees that the powers stabilize: $M_Y^2 = M_Y^3$; as soon as that happens one has the matrix of distances, d_Y . The powers will always stabilize if the set of vertices is finite; this is the only case we've been considering.⁶

⁶The method works even in the infinite case: one takes the infimum of all powers M_Y^n . The result always defines a Lawvere metric space.

- Exercise 2.75.* 1. Write down the matrix M_X , for X as in Eq. (2.17).
 2. Calculate M_X^2 , M_X^3 , and M_X^4 . Check that M_X^4 is what you got for the distance matrix in Exercise 2.37. \diamond

This procedure gives an algorithm for computing the \mathcal{V} -category presented by any \mathcal{V} -weighted graph using matrix multiplication.

2.6 Summary and further reading

This chapter we thought of elements of posets as describing resources, with the order detailing whether one resource could be obtained from another. This naturally led to the question of how to describe what could be built from a pair of resources, which led us to consider monoid structures on posets. More abstractly, these monoidal posets were seen to be examples of enriched categories, or \mathcal{V} -categories, over the symmetric monoidal poset **Bool**. Changing **Bool** to the symmetric monoidal poset **Cost**, we arrived upon Lawvere metric spaces, a slight generalization of the usual notion of metric space. In terms of resources, these now detail the cost or effort of obtaining one resource from another.

At this point, we sought to get a better feel for \mathcal{V} -categories in two ways. First, by various important constructions: base change, functors, products. Second, by understanding how to present \mathcal{V} -categories using labelled graphs; here, perhaps surprisingly, we saw that matrix multiplication gives an algorithm to compute the hom-objects from a labelled graph.

Resource theories are discussed in much more detail in [CFS16; Fri17]. They provide many more examples of resource theories in science, including in thermodynamics, Shannon’s theory of communication channels, and quantum entanglement. They also discuss more of the numerical theory than we did, including calculating the asymptotic rate of conversion from one resource into another.

Enrichment is a fundamental notion in category theory, and we will we return to it in Chapter 4, generalizing the definition so that categories, rather than mere posets, can serve as bases of enrichment. In this more general setting we can still perform the constructions we introduced in Section 2.4—base change, functors, products—and many others; the authoritative, but by no means easy, reference on this is the book by the Kelly [Kel05].

While posets were familiar before category theory, Lawvere metric spaces are a beautiful observation due to, well, Lawvere; a deeper exploration than the taste we gave here can be found in his classic paper [Law73].

We observed that while we may use any symmetric monoidal poset as a base for enrichment, certain posets—quantales—are better than others. Quantales are well known for links to other parts of mathematics too. The word quantale is in fact a portmanteau of ‘quantum locale’, where quantum refers to quantum physics, and locale

is a fundamental structure in topology. For a book-length introduction of quantales and their applications, one might check [Ros90].

Note that while we have only considered commutative quantales, the noncommutative variety also arise naturally. For example, the powerset of any monoid forms a quantale that is commutative iff the monoid is. Another example is the set of all binary relations on a set X , where multiplication is relational composition; this is non-commutative. Such noncommutative quantales have application to concurrency theory, and in particular process semantics and automata; see [AV93] for details.

Databases: Categories, functors, and universal constructions

3.1 What is a database?

Integrating data from disparate sources is a major problem in industry today. A study in 2008 [BH08] showed that data integration accounts for 40% of IT (information technology) budgets, and that the market for data integration software was \$2.5 billion in 2007 and increasing at a rate of more than 8% per year. In other words, it is a major problem; but what is it?

A database is a system of interlocking tables Data becomes information when it is stored *in* a given *formation*. That is, the numbers and letters don't mean anything until they are organized, often into a system of interlocking tables. An organized system of interlocking tables is called a database. Here is a favorite example:

Employee	FName	WorksIn	Mgr	Department	DName	Secr
1	Alan	101	2	101	Sales	1
2	Ruth	101	2	102	IT	3
3	Kris	102	3			

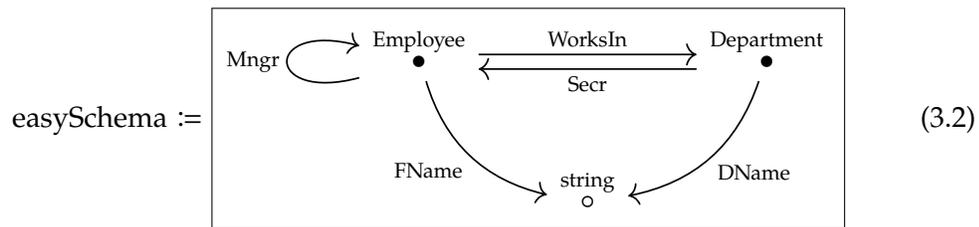
(3.1)

These two tables interlock by use of a special left-hand column, demarcated by a vertical line; it is called the ID column. The ID column of the first table is called "Employee", and the ID column of the second table is called "Department". The entries in the ID column—e.g. 1,2,3 or 101, 102—are like row labels; they indicate a whole row of the table they're in. Thus each row label must be unique (no two rows can have the same label), so that it can unambiguously specify its row.

Each table's ID column, and the unique identifiers found therein, is what allows for the interlocking mentioned above. Indeed, other entries in various tables can reference rows in a given table by use of its ID column. For example, each entry in the WorksIn column references a department for each employee; each entry in the Mngr (manager) column references an employee for each employee, and each entry in the Secr (secretary) column references an employee for each department. Managing all this cross-referencing is the purpose of databases.

Looking back at Eq. (3.1), one might notice that every non-ID column, found in either table, is a reference to a label of some sort. Some of these, namely WorksIn, Mngr, and Secr, are *internal references*; they refer to rows in some ID column. Others, namely FName and DName, are *external references*; they refer to strings or integers, which are also labels whose meaning is known more broadly. Internal reference labels can be changed as long as the change is consistent—1 could be replaced by 1001 everywhere without changing the meaning—whereas external reference labels certainly cannot! Changing Ruth to Bruce everywhere would change how people understood the data.

The reference structure for a given database—i.e. how tables interlock—tells us something about what information was intended to be stored in it. One may visualize the reference structure for Eq. (3.1) graphically as follows:



This is a kind of “Hasse diagram for databases”, much like the Hasse diagrams for posets in Remark 1.23. How should you read it?

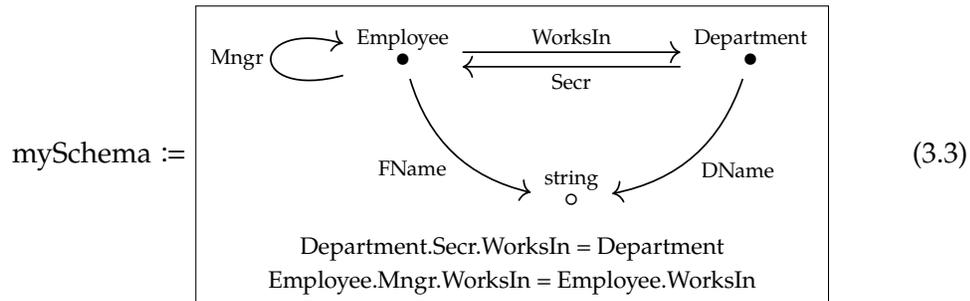
The two tables from Eq. (3.1) are represented in the graph (3.2) by the two black nodes, which are given the same name as the ID columns: Employee and Department. There is another node—drawn white rather than black—which represents the external reference type of strings, like “Alan”, “Alpha”, and “Sales”. The arrows in the diagram represent non-ID columns of the tables; they point in the direction of reference: WorksIn refers an employee to a department.

Exercise 3.1. Count the number of non-ID columns in Eq. (3.1). Count the number of arrows in Eq. (3.2). They should be the same number; is this a coincidence? \diamond

A Hasse-style diagram like the one in Eq. (3.2) can be called a *database schema*; it represents how the information is being organized, the formation in which the data is kept. One may add rules, sometimes called “business rules” to the schema, in order to ensure the integrity of the data. If these rules are violated, one knows that data being entered does not conform to the way the database designers intended. For example, the designers may enforce that

- every department's secretary must work in that department;

- every employee’s manager must work in the same department as the employee. Doing so changes the schema, say from “easySchema” (3.2) to “mySchema” below.



The difference is that easySchema plus constraints equals mySchema.

We will soon see that database schemas are categories \mathcal{C} , that the data itself is given by a “set-valued” functor $\mathcal{C} \rightarrow \mathbf{Set}$, and that databases can be mapped to each other via functors $\mathcal{C} \rightarrow \mathcal{D}$. In other words, there is a relatively large overlap between database theory and category theory. This has been worked out in a number of papers; see Section 3.6. It has also been implemented in working software, called FQL, which stands for *functorial query language*. Here is example FQL code for the schema shown above:

```

schema mySchema = {
  nodes
    Employee, Department;
  attributes
    DName : Department -> string,
    FName : Employee   -> string;
  arrows
    Mngr   : Employee   -> Employee,
    WorksIn : Employee   -> Department,
    Secr   : Department -> Employee;
  equations
    Department.Secr.worksIn = Department,
    Employee.Mngr.WorksIn   = Employee.WorksIn;
}

```

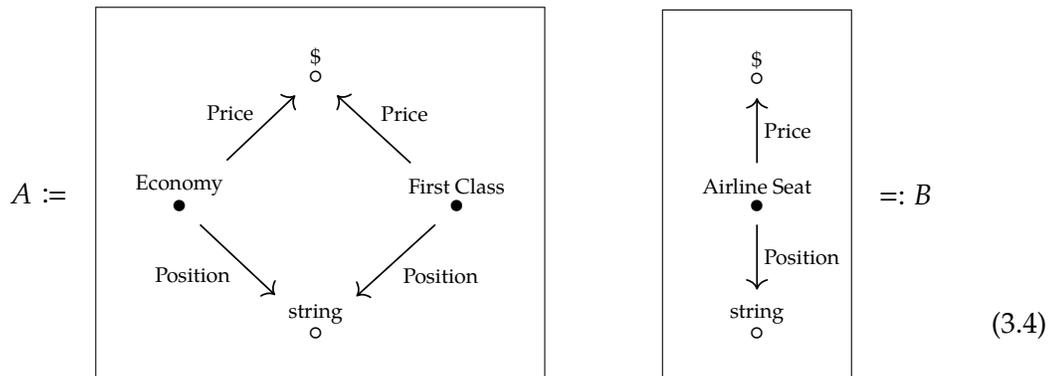
Communication between databases We have said that databases are designed to store information about something. But different people or organizations can view the same sort of thing in different ways. For example, one bank stores its financial records according to European standards and another does so according to Japanese standards. If these two banks merge into one, they will need to be able to share data despite differences in the shape of their database schemas.

Such problems are huge and intricate in general, because databases often comprise hundreds or thousands of interlocking tables. Moreover, these problems occur more frequently than just when companies want to merge. It is quite common that a given company moves data between databases on a daily basis. The reason is that different

ways of organizing information are convenient for different purposes. Just like we pack our clothes in a suitcase when traveling but use a closet at home, there is generally not one best way to organize anything.

Category theory provides a mathematical approach for translating between these different organizational forms. That is, it formalizes a sort of automated reorganizing process called *data migration*, which takes data that fits snugly in one schema and moves it into another.

Here is a simple case. Imagine an airline company has two different databases, perhaps created at different times, that hold roughly the same data.



Schema A has more detail than schema B —an airline seat may be in first class or economy—but they are roughly the same. We will see that they can be connected by a functor, and that data conforming to A can be migrated through this functor to schema B and vice versa.

The statistics at the beginning of this section show that this sort of problem—when occurring at enterprise scale—has proved difficult and expensive. If one attempts to move data from a source schema to a target schema, the migrated data could fail to fit into the target schema or fail to satisfy some of its constraints. This happens surprisingly often in the world of business: a night may be spent moving data, and the next morning it is found to have arrived broken and unsuitable for further use. In fact, it is believed that over half of database migration projects fail.

In this chapter, we will discuss a category-theoretic method for migrating data. Using categories and functors, one can prove up front that a given data migration will not fail, i.e. that the result is guaranteed to fit into the target schema and satisfy all its constraints.

The material in this chapter gets to the heart of category theory: in particular, we discuss categories, functors, natural transformations, adjunctions, limits, and colimits. In fact, many of these ideas have been present in the discussion above:

- The schema pictures, e.g. Eq. (3.3) depict categories C .
- The instances, e.g. Eq. (3.1) are functors from C to a certain category called **Set**.
- The implicit mapping in Eq. (3.4) that takes economy and first class seats in A to airline seats in B constitutes a functor $A \rightarrow B$.

- The notion of data migration for moving data between schemas is formalized by adjoint functors.

We begin in Section 3.2 with the definition of categories and a bunch of different sorts of examples. In Section 3.3 we bring back databases, in particular their instances and the maps between them, by discussing functors and natural transformations. In Section 3.4 we discuss data migration by way of adjunctions, which generalize the Galois connections we introduced in Section 1.5. Finally in Section 3.5 we give a bonus section on limits and colimits.¹

3.2 Categories

A category C consists of four pieces of data—objects, morphisms, identities, and a composition rule—satisfying two properties.

Definition 3.2. To specify a *category* C :

- (i) one specifies a collection $\text{Ob}(C)$, elements of which are called *objects*.
- (ii) for every two objects c, d , one specifies a set $C(c, d)$,² elements of which are called *morphisms* from c to d .
- (iii) for every object $c \in \text{Ob}(C)$, one specifies a morphism $\text{id}_c \in C(c, c)$, called the *identity morphism* on c .
- (iv) for every three objects $c, d, e \in \text{Ob}(C)$ and morphisms $f \in C(c, d)$ and $g \in C(d, e)$, one specifies a morphism $f.g \in C(c, e)$, called *the composite of f and g* .

It is convenient to denote elements $f \in C(c, d)$ as $f: c \rightarrow d$. Here, c is called the *domain* of f , and d is called the *codomain* of f .

These constituents are required to satisfy two conditions:

- (a) *unitality*: for any morphism $f: c \rightarrow d$, composing with the identities at c or d does nothing: $\text{id}_c . f = f$ and $f . \text{id}_d = f$.
- (b) *associativity*: for any three morphisms $f: c_0 \rightarrow c_1$, $g: c_1 \rightarrow c_2$, and $h: c_2 \rightarrow c_3$, the following are equal: $(f.g).h = f.(g.h)$. We can write it simply as $f.g.h$.

Our next goal is to give lots of examples of this concept. Our first source of examples is that of free and finitely presented categories, which generalize the notion of Hasse diagram from Remark 1.23.

3.2.1 Free categories

Recall from Definition 1.22 that a graph consists of two types of thing: vertices and arrows. From there one can define paths, which are just head-to-tail sequences of arrows. Every path p has a start vertex and an end vertex; if p goes from v to w ,

¹By “bonus”, we mean that this is important material—limits and colimits will show up throughout the book and throughout ones interaction with category theory—so we think the reader will especially benefit from this material in the long run.

²This set $C(c, d)$ is often denoted $\text{Hom}_C(c, d)$, and called the “hom-set from c to d .” The word “hom” stands for homomorphism, of which the word “morphism” is a shortened version.

we write $p: v \rightarrow w$. To every vertex v , there is a trivial path, containing no arrows, starting and ending at the given vertex; we often denote it simply by v . We may also concatenate paths: if $p: v \rightarrow w$ and $q: w \rightarrow x$, their concatenation is denoted $p.q$, and it goes $v \rightarrow x$.

In Chapter 1, we used graphs to depict posets (V, \leq) : the vertices form the elements of the poset and for the order, we say that $v \leq w$ if there is a path from v to w in G . We will now use graphs in a very similar way to depict certain categories, known as *free categories*. Then we will explain the strong relationship between posets and categories in Section 3.2.3.

For any graph $G = (V, A, s, t)$, we can define a category $\mathbf{Free}(G)$, called the *free category on G* , whose objects are the vertices V and whose morphisms from c to d are the paths from c to d . The identity morphism on an object c is simply the trivial path at c . Composition is given by concatenation of paths.

For example, we define $\mathbf{2}$ to be the free category generated by the graph shown below:

$$\mathbf{2} := \mathbf{Free} \left(\boxed{\begin{array}{ccc} v_1 & \xrightarrow{f_1} & v_2 \\ \bullet & & \bullet \end{array}} \right) \quad (3.5)$$

It has two objects v_1 and v_2 , and three morphisms: $\text{id}_{v_1}: v_1 \rightarrow v_1$, $f_1: v_1 \rightarrow v_2$, and $\text{id}_{v_2}: v_2 \rightarrow v_2$. Here id_{v_1} is the path of length 0 starting and ending at v_1 , f_1 is the path of length 1 consisting of just the arrow f_1 , and id_{v_2} is the length 0 path at v_2 . As our notation suggests id_{v_1} is the identity morphism for the object v_1 , and similarly id_{v_2} for v_2 . As composition is given by concatenation, we have, for example $\text{id}_{v_1} \cdot f_1 = f_1$, $\text{id}_{v_2} \cdot \text{id}_{v_2} = \text{id}_{v_2}$, and so on.

From now on, we may elide the difference between a graph and the corresponding free category $\mathbf{Free}(G)$, at least when the one we mean is clear enough from context.

Exercise 3.3. For $\mathbf{Free}(G)$ to really be a category, we must check that this data we specified obeys the unitality and associativity properties. Check that these are obeyed. \diamond

Exercise 3.4. The free category on the graph shown here:³

$$\mathbf{3} := \mathbf{Free} \left(\boxed{\begin{array}{ccccc} v_1 & \xrightarrow{f_1} & v_2 & \xrightarrow{f_2} & v_3 \\ \bullet & & \bullet & & \bullet \end{array}} \right) \quad (3.6)$$

has three objects and six morphisms: the three vertices and six paths in the graph.

Create six names, one for each of the six morphisms in $\mathbf{3}$. Write down a six-by-six table, label the rows and columns by the six names you chose.

1. Fill out the table by writing the name of the composite in each cell.
2. Where are the identities? \diamond

Exercise 3.5. Let's make some definitions, based on the pattern above:

1. What is the category $\mathbf{1}$? That is, what are its objects and morphisms?
2. What is the category $\mathbf{0}$?
3. What is the formula for the number of morphisms in \mathbf{n} for arbitrary $n \in \mathbb{N}$? \diamond

³As mentioned above, we elide the difference between the graph and the corresponding free category.

Example 3.6 (Natural numbers as a free category). Consider the following graph:



It has only one vertex and one arrow, but it has infinitely many paths. Indeed, it has a unique path of length n for every natural number $n \in \mathbb{N}$. That is, $\text{Path} = \{z, s, s.s, s.s.s, \dots\}$, where we write z for the length 0 path on z ; it represents the morphism id_z . There is a one-to-one correspondence between Path and the natural numbers, $\mathbb{N} = \{0, 1, 2, 3, \dots\}$.

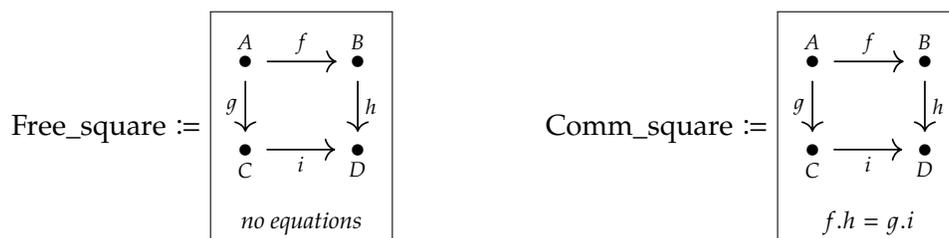
This is an example of a category with one object. Such categories are known as *monoids*, because they are defined by a set (of morphisms) with an identity and a single binary operation (the composition rule). Monoidal posets, which we met in Chapter 2, are so-named because they add monoid structure to posets. \diamond

Exercise 3.7. In Example 3.6 we identified the paths of the loop graph (3.7) with numbers $n \in \mathbb{N}$. Paths can be concatenated. Given numbers $m, n \in \mathbb{N}$, what number corresponds to the concatenation of their associated paths? \diamond

3.2.2 Presenting categories via path equations

So for any graph G , there is a free category on G . But we don't have to stop there: we can add equations between paths in the graph, and still get a category. We are only allowed to equate two paths p and q when they are *parallel*, meaning they have the same source vertex and the same target vertex.

A finite graph with path equations is called a *finite presentation* for a category, and the category that results is known as a *finitely presented category*. Here are two examples:



Both of these are presentations of categories: in the left-hand one, there are no equations so it presents a free category, as discussed in Section 3.2.1. The free square category has ten morphisms, because every path is a unique morphism.

Exercise 3.8. 1. Write down the ten paths in the free square category above.
2. Name two different paths that are parallel.
3. Name two different paths that are not parallel. \diamond

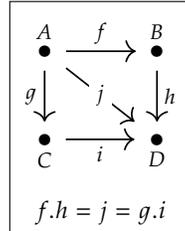
On the other hand, the category presented on the right has only nine morphisms, because $f.h$ and $g.i$ are made equal. This category is called the “commutative square”.

Its morphisms are

$$\{A, B, C, D, f, g, h, i, f.h\}$$

One might say “the missing one is $g.i$,” but that is not quite right: $g.i$ is there too, because it is equal to $f.h$. As usual, A denotes id_A , etc.

Exercise 3.9. Write down all the morphisms in the category presented by the following diagram:



◇

Example 3.10. Here are two more another examples:



The set of morphisms in C is $\{z, s\}$, where $s.s = z$. Thus $s.s.s = s$ and $s.s.s.s = z$, etc.

◆

Exercise 3.11. Write down all the morphisms in the category \mathcal{D} from Example 3.10. ◇

Remark 3.12. We can now see that the schemas in Section 3.1, e.g. Eqs. (3.2) and (3.3) are finite presentations of categories. We will come back to that in Section 3.3.

3.2.3 Posets and free categories: two ends of a spectrum

Now that we have used graphs to depict posets in Chapter 1 and categories above, one may want to know the relationship between these uses. The main idea we want to explain now is that

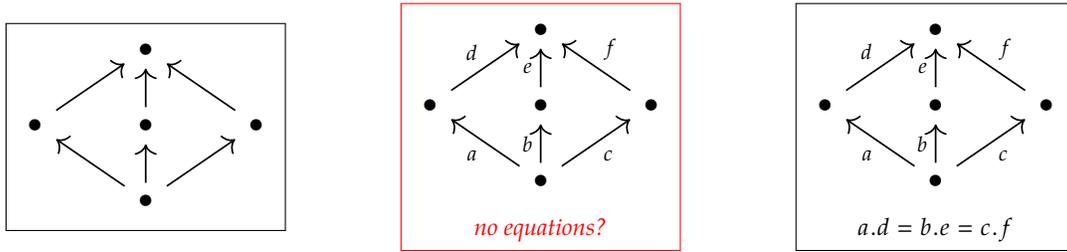
“A poset is a category where every two parallel arrows are the same.”

Thus any poset can be regarded as a category, and any category can be somehow “crushed down” into a poset. Let’s discuss these ideas.

Posets as categories Suppose (P, \leq) is a poset. It specifies a category \mathcal{P} as follows. The objects of \mathcal{P} are precisely the elements of P ; that is, $\text{Ob}(\mathcal{P}) = P$. As for morphisms, \mathcal{P} has exactly one morphism $p \rightarrow q$ if $p \leq q$ and no morphisms $p \rightarrow q$ if $p \not\leq q$.

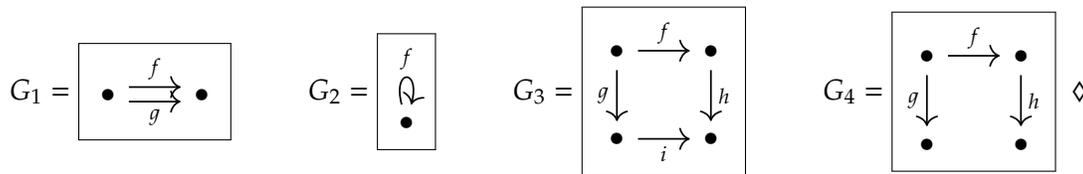
This means that a Hasse diagram for a poset can be thought of a presentation of a category where, for all vertices p and q , every two paths from $p \rightarrow q$ are declared

equal. For example, in Eq. (1.2) we saw a Hasse diagram that was like the graph on the left:



The Hasse diagram (left) might look the most like the free category presentation (middle) which has no equations, but that is not correct. The free category has three morphisms from bottom object to top object, whereas posets are categories with at most one morphism between two given objects. Thus the diagram on the right is the correct presentation for the poset on the left.

Exercise 3.13. What equations would you need to add to the graphs below in order to present the associated posets?



The poset reflection of a category Given any category C , one can obtain a poset (C, \leq) from it by destroying the distinction between any two parallel morphisms. That is, let $C = \text{Ob}(C)$, and put $c_1 \leq c_2$ iff $C(c_1, c_2) \neq \emptyset$. If there is one, or two, or fifty, or infinitely many morphisms $c_1 \rightarrow c_2$ in C , the poset reflection does not see the difference. But it does see the difference between some and none.

Exercise 3.14. What is the poset reflection of the category \mathbb{N} from Example 3.6? \diamond

We have only discussed adjoint functors between posets, but soon we will discuss adjoints in general. Here is a statement you might not understand exactly, but it's true; you can ask a category theory expert about it and they should be able to explain it to you:

Considering a poset as a category is right adjoint to turning a category into a poset by poset reflection.

Remark 3.15 (Ends of a spectrum). The main point of this subsection is that both posets and free categories are specified by a graph without path equations, but they denote opposite ends of a spectrum. In both cases, the vertices of the graph become the objects of a category and the paths become morphisms. But in the case of free categories, each path becomes a different morphism. In the case of posets, all parallel paths become the same morphism. Every category presentation, i.e. graph with equations, lies somewhere in between the free category and its poset reflection.

3.2.4 Important categories in mathematics

We have been talking about category presentations, but there are categories that are best understood directly, not by way of presentations. Recall the definition of category from Definition 3.2. The most important category in mathematics is the category of sets.

Definition 3.16. The *category of sets*, denoted **Set**, is defined as follows.

- (i) $\text{Ob}(\mathbf{Set})$ is the collection of all sets.
- (ii) If S and T are sets, then $\mathbf{Set}(S, T) = \{f: S \rightarrow T \mid f \text{ is a function}\}$.
- (iii) For each set S , the identity morphism is the function $\text{id}_S: S \rightarrow S$ given by $\text{id}_S(s) = s$ for each $s \in S$.
- (iv) Given $f: S \rightarrow T$ and $g: T \rightarrow U$, their composite $f.g$ sends $s \in S$ to $g(f(s)) \in U$.

These definitions satisfy the unitality and associativity conditions, so **Set** is indeed a category.

Closely related is the category **FinSet**. This is the category where the objects are finite sets and the morphisms are functions between them.

Exercise 3.17. Let $\underline{2} = \{1, 2\}$ and $\underline{3} = \{1, 2, 3\}$. These are objects in the category **Set** discussed in Definition 3.16. Write down all the elements of the set $\mathbf{Set}(\underline{2}, \underline{3})$; there should be nine. \diamond

Remark 3.18. You may have wondered what categories have to do with \mathcal{V} -categories (Definition 2.28); perhaps you think the definitions hardly look alike. Despite the term ‘enriched category’, \mathcal{V} -categories are not categories with extra structure. While some sorts of \mathcal{V} categories, such as **Bool**-categories, i.e. posets, can naturally be seen as categories, other sorts, such as **Cost**-categories, cannot.

The reason for the importance of **Set** is that, if we generalize the definition of enriched category (Definition 2.28), we find that categories in the sense of Definition 3.2 are exactly **Set**-categories—so categories are \mathcal{V} -categories for a very special choice of \mathcal{V} . We’ll come back to this in Section 4.4.4. For now, we simply remark that just like a deep understanding of the category **Cost**—for example, knowing that it is a quantale—yields insight into Lawvere metric spaces, so the study of **Set** yields insights into categories.

There are many other categories that mathematicians care about:

- **Top**: the category of topological spaces (neighborhoods)
- **Grph**: the category of graphs (connection)
- **Meas**: the category of measure spaces (amount)
- **Mon**: the category of monoids (action)
- **Grp**: the category of groups (reversible action, symmetry)
- **Cat**: the category of small categories (action in context, structure)

But in fact, this does not at all do justice to which categories mathematicians think about. They work with whatever category they find fits their purpose at the time, like ‘the category of Riemannian manifolds of dimension at most 4’.

Here is one more source of examples: take any category you already have and reverse all its morphisms; the result is again a category.

Definition 3.19. Let C be a category. Its *opposite*, denoted C^{op} is the category with the same objects, $\text{Ob}(C^{\text{op}}) = \text{Ob}(C)$, and for any two objects $c, d \in \text{Ob}(C)$, one has $C^{\text{op}}(c, d) = C(d, c)$. Identities and composition are as in C .

3.2.5 Isomorphisms in a category

The previous sections have all been about examples of categories: free categories, presented categories, and important categories in math. In this section, we briefly switch gears and talk about an important concept in category theory, namely the concept of isomorphism.

In a category, there is often the idea that two objects are interchangeable. For example, in the category **Set**, one can exchange the set $\{\blacksquare, \square\}$ for the set $\{0, 1\}$ and everything will be the same, other than the names for the elements. Similarly, if one has a poset with elements a, b , such that $a \leq b$ and $b \leq a$, i.e. $a \cong b$, then are essentially the same. How so? Well they act the same, in that for any other object c , we know that $c \leq a$ if and only if $c \leq b$, and $c \geq a$ iff $c \geq b$. The notion of isomorphism formalizes this notion of interchangeability.

Definition 3.20. An *isomorphism* is a morphism $f: A \rightarrow B$ such that there exists a morphism $g: B \rightarrow A$ satisfying $f.g = \text{id}_A$ and $g.f = \text{id}_B$. In this case we call f and g *inverses*, and we often write $g = f^{-1}$, or equivalently $f = g^{-1}$. We also say that A and B are *isomorphic* objects.

Example 3.21. The set $A := \{a, b, c\}$ and the set $\underline{3} = \{1, 2, 3\}$ are isomorphic; that is, there exists an isomorphism $f: A \rightarrow \underline{3}$ given by $f(a) = 2, f(b) = 1, f(c) = 3$. \blacklozenge

Exercise 3.22. 1. What is the inverse $f^{-1}: \underline{3} \rightarrow A$ of the function f given in Example 3.21? \blacklozenge

2. How many distinct isomorphisms are there $A \rightarrow \underline{3}$? \blacklozenge

Exercise 3.23. Show that in any category C , for any object $c \in C$, the identity id_c is an isomorphism. \blacklozenge

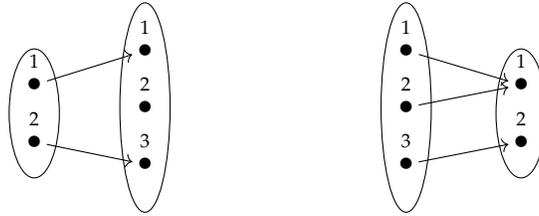
Exercise 3.24. Recall Examples 3.6 and 3.10. A monoid in which every morphism is an isomorphism is known as a *group*.

1. Is the monoid in Example 3.6 a group? \blacklozenge
2. What about the monoid in Example 3.10? \blacklozenge

Exercise 3.25. Let G be a graph, and let $\mathbf{Free}(G)$ be the corresponding free category. Somebody tells you that the only isomorphisms in $\mathbf{Free}(G)$ are the identity morphisms. Is that person correct? Why or why not? \blacklozenge

Example 3.26. In this example, we will see that it is possible for g and f to be almost—but not quite—inverses, in a certain sense.

Consider the functions $f: \underline{2} \rightarrow \underline{3}$ and $g: \underline{3} \rightarrow \underline{2}$ drawn below:



Then the reader should be able to instantly check that $f.g = \text{id}_{\underline{2}}$ but $g.f \neq \text{id}_{\underline{3}}$. Thus f and g are not inverses and hence not isomorphisms. \blacklozenge

3.3 Functors, natural transformations, and databases

In Section 3.1 we showed some database schemas: graphs with path equations. Then in Section 3.2.2 we said that graphs with path equations correspond to finitely presented categories. Now we want to explain what the data in a database is, as a way to introduce functors. To do so, we begin by noticing that sets and functions—the objects and morphisms in the category **Set**—can be captured by simple databases.

3.3.1 Sets and functions as databases

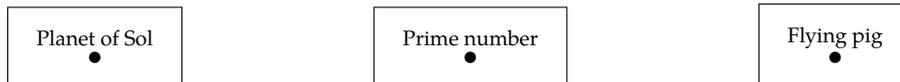
The first observation is that any set can be understood as a table with only one column: the ID column.

Planet of Sol	Prime number	Flying pig
Mercury	2	
Venus	3	
Earth	5	
Mars	7	
Jupiter	11	
Saturn	13	
Uranus	17	
Neptune	⋮	

Rather than put the elements of the set between braces, e.g. $\{2, 3, 5, 7, 11, \dots\}$, we write them down as rows in a table.

In databases, single-column tables are often called controlled vocabularies, or master data. Now to be honest, we can only write out every single entry in a table when its set of rows is finite. A database practitioner might find the idea of our prime number table a bit unrealistic. But we're mathematicians, so since the idea makes perfect sense abstractly, we will continue to think of sets as one-column tables.

The above databases have schemas consisting of just one vertex:



Obviously, there's really not much difference between these schemas, other than the label of the unique vertex. So we could say "sets are databases whose schema consists of a single vertex". Let's move on to functions.

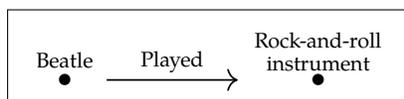
A function $f: A \rightarrow B$ can almost be depicted as a two-column table

Beatle	Played
George	Lead guitar
John	Rhythm guitar
Paul	Bass guitar
Ringo	Drums

except it is unclear whether the elements of the right-hand column exhaust all of B . What if there are rock-and-roll instruments out there that none of the Beatles played? So a function $f: A \rightarrow B$ requires two tables, one for A and its f column, and one for B :

Beatle	Played	Rock-and-roll instrument
George	Lead guitar	Bass guitar
John	Rhythm guitar	Drums
Paul	Bass guitar	Keyboard
Ringo	Drums	Lead guitar
		Rhythm guitar

Thus the database schema for any function is just a labeled version of 2:



The lesson is that an instance of a database takes a presentation of a category, and turns every vertex into a set, and every arrow into a function. As such, it describes a map from the presented category to the category **Set**. In Section 2.4.2 we saw that maps of \mathcal{V} -categories are known as \mathcal{V} -functors. Similarly, we call maps of plain old categories, functors.

3.3.2 Functors

A functor is a mapping between categories. It sends objects to objects and morphisms to morphisms, all while preserving identities and composition. Here is the formal definition.

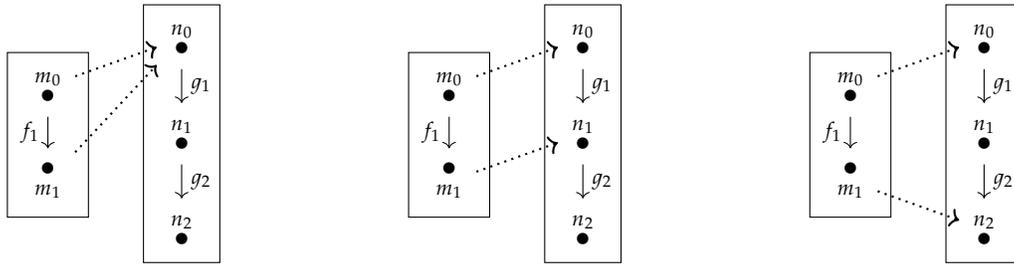
Definition 3.27. Let C and \mathcal{D} be categories. To specify a *functor from C to \mathcal{D}* , denoted $F: C \rightarrow \mathcal{D}$,

- (i) for every object $c \in \text{Ob}(C)$, one specifies an object $F(c) \in \text{Ob}(\mathcal{D})$;
- (ii) for every morphism $f: c_1 \rightarrow c_2$ in C , one specifies a morphism $F(f): F(c_1) \rightarrow F(c_2)$ in \mathcal{D} .

The above constituents must satisfy two properties:

- (a) for every object $c \in \text{Ob}(C)$, we have $F(\text{id}_c) = \text{id}_{F(c)}$.
- (b) for every three objects $c_1, c_2, c_3 \in \text{Ob}(C)$ and two morphisms $f \in C(c_1, c_2)$, $g \in C(c_2, c_3)$, the equation $F(f.g) = F(f).F(g)$ holds in \mathcal{D} .

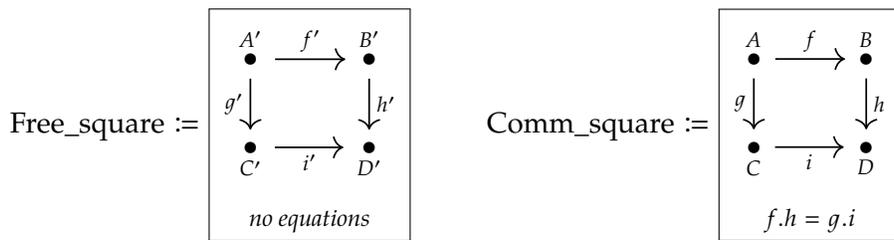
Example 3.28. For example, here we draw three functors $F: \mathbf{2} \rightarrow \mathbf{3}$:



In each case, the dotted arrows show what the functor F does to the vertices in $\mathbf{2}$; once that information is specified, it turns out—in this special case—that what F does to the three paths in $\mathbf{2}$ is completely determined. In the left-hand diagram, F sends every path to the trivial path, i.e. the identity on n_0 . In the middle diagram $F(m_0) = n_0$, $F(f_1) = g_1$, and $F(m_1) = n_1$. In the right-hand diagram, $F(m_0) = n_0$, $F(m_1) = n_2$, and $F(f_1) = g_1.g_2$. \blacklozenge

Exercise 3.29. Above we wrote down three functors $\mathbf{2} \rightarrow \mathbf{3}$. Find and write down all the rest of the functors $\mathbf{2} \rightarrow \mathbf{3}$. \blacklozenge

Example 3.30. Recall the categories presented by `Free_square` and `Comm_square` in Section 3.2.2. Here they are again, with ' added to the labels in `Free_square` to help distinguish them:



There are lots of functors from the free square category (let's call it \mathcal{F}) to the commutative square category (let's call it \mathcal{C}).

However, there is exactly one such functor $F: \mathcal{F} \rightarrow \mathcal{C}$ that sends A' to A , B' to B , C' to C , and D' to D . That is, once we have made this decision about F on objects, each of the ten paths in \mathcal{F} is forced to go to a certain path in \mathcal{C} : the one with the right source and target. \blacklozenge

Exercise 3.31. Say where each morphism in \mathcal{F} is sent under the above functor F . \blacklozenge

All of our example functors so far have been completely determined by what they do on objects, but this is usually not the case.

Exercise 3.32. Consider the free categories $\mathcal{C} = \boxed{\bullet \rightarrow \bullet}$ and $\mathcal{D} = \boxed{\bullet \rightrightarrows \bullet}$. Give two functors $F, G: \mathcal{C} \rightarrow \mathcal{D}$ that act the same on objects but differently on morphisms. \diamond

Example 3.33. There are also lots of functors from the commutative square category \mathcal{C} to the free square category \mathcal{F} , but *none* that sends A to A' , B to B' , C to C' , and D to D' . The reason is that if F were such a functor, then since $f.h = g.i$ in \mathcal{F} , we would have $F(f.h) = F(g.i)$, but then the rules of functors would let us reason as follows:

$$f'.h' = F(f).F(h) = F(f.h) = F(g.i) = F(g).F(i) = g'.i'$$

The resulting equation, $f'.h' = g'.i'$ does not hold in \mathcal{F} because it is a free category (there are “no equations”): every two paths are considered different morphisms. Thus our proposed F is not a functor. \diamond

Example 3.34 (Functors between posets are monotone maps). Recall from Section 3.2.3 that posets are categories with at most one morphism between any two objects. A functor between posets is exactly a monotone map.

For example, consider the poset (\mathbb{N}, \leq) considered as a category \mathcal{N} with objects $\text{Ob}(\mathcal{N}) = \mathbb{N}$ and a unique morphism $m \rightarrow n$ iff $m \leq n$. A functor $F: \mathcal{N} \rightarrow \mathcal{N}$ sends each object $n \in \mathbb{N}$ to an object $F(n) \in \mathbb{N}$. It must send morphisms in \mathcal{N} to morphisms in \mathbb{N} . This means if there is a morphism $m \rightarrow n$ then there had better be a morphism $F(m) \rightarrow F(n)$. In other words, if $m \leq n$, then we had better have $F(m) \leq F(n)$. But as long as $m \leq n$ implies $F(m) \leq F(n)$, we have a functor.

Thus a functor $F: \mathcal{N} \rightarrow \mathcal{N}$ and a monotone map $\mathbb{N} \rightarrow \mathbb{N}$ are the same thing. \diamond

Exercise 3.35 (The category of categories). Back in the primordial ooze, there is a category **Cat** in which *the objects are themselves categories*. Your task here is to construct this category.

1. Given any category \mathcal{C} , show that there exists a functor $\text{id}_{\mathcal{C}}: \mathcal{C} \rightarrow \mathcal{C}$, known as the *identity functor on \mathcal{C}* , that maps each object to itself and each morphism to itself.
2. Note that a functor $\mathcal{C} \rightarrow \mathcal{D}$ consists of a function from $\text{Ob}(\mathcal{C})$ to $\text{Ob}(\mathcal{D})$ and for each pair of objects $c_1, c_2 \in \mathcal{C}$ a function from $\mathcal{C}(c_1, c_2)$ to $\mathcal{D}(F(c_1), F(c_2))$.
3. Show that given $F: \mathcal{C} \rightarrow \mathcal{D}$ and $G: \mathcal{D} \rightarrow \mathcal{E}$, we can define a new functor $F.G: \mathcal{C} \rightarrow \mathcal{E}$ just by composing functions.
4. Show that there is a category, call it **Cat**, where the objects are categories, morphisms are functors, and identities and composition are given as above. \diamond

3.3.3 Databases as Set-valued functors

Let \mathcal{C} be a category, and recall the category **Set** from Definition 3.16. A functor $F: \mathcal{C} \rightarrow \mathbf{Set}$ is known as a *set-valued functor* on \mathcal{C} . Much of database theory (not how to make them fast, but what they are and what you do with them) can be cast in this light.

Indeed, we already saw in Remark 3.12 that any database schema can be regarded as (presenting) a small category C . The next thing to notice is that the data itself—any instance of the database—is given by set-valued functor $I: C \rightarrow \mathbf{Set}$. The only additional detail is that for any white node, such as $c = \overset{\text{string}}{\circ}$, we want to force I to map to the set of strings. We suppress this detail in the following definition.

Definition 3.36. Let C be a schema, i.e. a finitely presented category. A C -instance is a functor $I: C \rightarrow \mathbf{Set}$.⁴

Exercise 3.37. Let $\mathbf{1}$ denote the category with one object, called 1 , one identity morphism id_1 , and no other morphisms. For any functor $F: \mathbf{1} \rightarrow \mathbf{Set}$ one can extract a set $F(1)$. Show that for any set S , there is a functor $F_S: \mathbf{1} \rightarrow \mathbf{Set}$ such that $F_S(1) = S$. \diamond

The above exercise reaffirms that the set of planets, the set of prime numbers, and the set of flying pigs are all set-valued functors—instances—on the schema $\mathbf{1}$. Similarly, set-valued functors on the category $\mathbf{2}$ are functions. All our examples so far are for the situation where the schema is a free category (no equations). Let's try an example of a category that is not free.

Example 3.38. Consider the following category:

$$C := \boxed{\begin{array}{c} s \\ \curvearrowright \\ \bullet \\ z \\ s.s = s \end{array}} \quad (3.8)$$

What is a set-valued functor $F: C \rightarrow \mathbf{Set}$? It will consist of a set $Z := F(z)$ and a function $S := F(s): Z \rightarrow Z$, subject to the requirement that $S.S = S$. Here are some examples

- Z is the set of US citizens, and S sends each citizen to her or his president. The president's president is her- or him-self.
- $Z = \mathbb{N}$ is the set of natural numbers and S sends each number to 0. In particular, 0 goes to itself.
- Z is the set of all well-formed arithmetic expressions, such as $13 + (2 * 4)$ or -5 , that one can write using integers and the symbols $+$, $-$, $*$, $(,)$. The function S evaluates the expression to return an integer, which is itself a well-formed expression. The evaluation of an integer is itself.
- $Z = \mathbb{N}_{\geq 2}$, and S sends n to its smallest prime factor. The smallest prime factor of a prime is itself.

⁴Warning: a C -instance is a state of the database “at an instant in time”. The term “instance” should not be confused with that in object oriented programming, which would correspond more to what we call a row $r \in I(c)$.

$\mathbb{N}_{\geq 2}$	smallest prime factor
2	2
3	3
4	2
\vdots	\vdots
49	7
50	2
51	3
\vdots	\vdots

Exercise 3.39. Above, we thought of the sort of data that would make sense for the schema (3.8). Give an example of the sort of data that would make sense for the

following schemas:

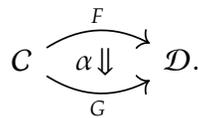
1. $\begin{matrix} s \\ \downarrow \\ \bullet \\ z \\ s.s = z \end{matrix}$

2. $\begin{matrix} a & \xrightarrow{f} & b & \begin{matrix} \xrightarrow{g} \\ \xrightarrow{h} \end{matrix} & c \\ & & & & \\ & & & & f.g = f.h \end{matrix}$

The main idea is this: a database schema is a category, and an instance on that schema—the data itself—is a set-valued functor. All the constraints, or business rules, are ensured by the rules of functors, namely that functors preserve composition.

3.3.4 Natural transformations

If C is a schema—i.e. a finitely presented category—then there are many database instances on it, which we can organize into a category. But this is part of a larger story, namely that of natural transformations. An abstract picture to have in mind is this:



Definition 3.40. Let C and \mathcal{D} be categories, and let $F, G: C \rightarrow \mathcal{D}$ be functors. To specify a natural transformation $\alpha: F \Rightarrow G$,

- (i) for each object $c \in C$, one specifies a morphism $\alpha_c: F(c) \rightarrow G(c)$ in \mathcal{D} , called the *c-component of α* .

These components must satisfy the following, called the *naturality condition*:

- (a) for every morphism $f: c \rightarrow d$ in C , the following equation must hold:

$$I(f). \alpha_d = \alpha_c. J(f).$$

A natural transformation $\alpha: F \rightarrow G$ is called a *natural isomorphism* if each component α_c is an isomorphism in \mathcal{D} .

The naturality condition can also be written as a so-called *commutative diagram*. A diagram in a category is drawn as a graph whose vertices and arrows are labeled by

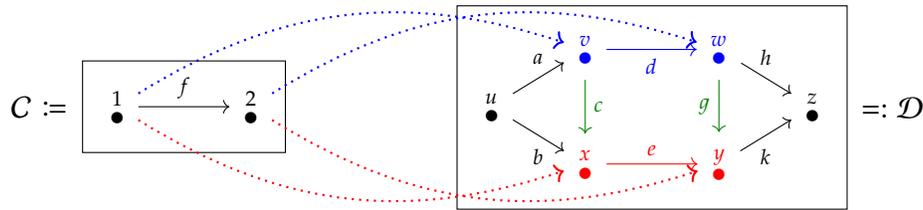
objects and morphisms in the category. For example, here is a diagram that's relevant to the naturality condition in Definition 3.40:

$$\begin{array}{ccc}
 I(c) & \xrightarrow{\alpha_c} & J(c) \\
 I(f) \downarrow & & \downarrow J(f) \\
 I(d) & \xrightarrow{\alpha_d} & J(d)
 \end{array} \tag{3.9}$$

Definition 3.41. A *diagram* in \mathcal{C} is a functor from any category $\mathcal{D}: \mathcal{J} \rightarrow \mathcal{C}$. We say that the diagram D *commutes* if $Df = Df'$ holds for every parallel pair of morphisms $f, f': a \rightarrow b$ in \mathcal{J} .⁵ We call \mathcal{J} the *indexing category* for the diagram.

In terms of Eq. (3.9), the only case of two parallel morphisms is that of $I(c) \rightrightarrows J(d)$, so to say that the diagram commutes is to say that $I(f) \cdot \alpha_d = \alpha_c \cdot J(f)$. This is exactly the naturality condition from Definition 3.40.

Example 3.42. A representative picture is as follows:



We have depicted, in blue and red respectively, two functors $F, G: \mathcal{C} \rightarrow \mathcal{D}$. A natural transformation $\alpha: F \Rightarrow G$ is given by choosing components $\alpha_1: v \rightarrow x$ and $\alpha_2: w \rightarrow y$; we have highlighted the only choice for each in green.

The key point is that the functors F and G are ways of viewing the category \mathcal{C} as lying inside the category \mathcal{D} . The natural transformation α , then, is a way of relating these two views using the morphisms in \mathcal{D} . \blacklozenge

Example 3.43. We said in Exercise 3.37 that a functor $\mathbf{1} \rightarrow \mathbf{Set}$ can be identified with a set. So suppose A and B are sets considered as functors $A, B: \mathbf{1} \rightarrow \mathbf{Set}$. A natural transformation between these functors is just a function between the sets. \blacklozenge

Definition 3.44. Let \mathcal{C} and \mathcal{D} be categories. We denote by $\mathcal{C}^{\mathcal{D}}$ the category whose objects are functors $F: \mathcal{D} \rightarrow \mathcal{C}$ and whose morphisms $\mathcal{C}^{\mathcal{D}}(F, G)$ are the natural transformations $\alpha: F \rightarrow G$. This category $\mathcal{C}^{\mathcal{D}}$ is called the *functor category*.

Exercise 3.45. Let's look more deeply at how $\mathcal{D}^{\mathcal{C}}$ is a category.

1. Figure out how to compose natural transformations. (Hint: an expert tells you "for each object $c \in \mathcal{C}$, compose the c -components".)
2. Propose an identity natural transformation on any object $F \in \mathcal{D}^{\mathcal{C}}$, and check that it is unital. \blacklozenge

⁵We package this formally by saying that D commutes iff it factors through the poset reflection of \mathcal{J} .

Example 3.46. In our new language, Example 3.43 says that \mathbf{Set}^1 is equivalent to \mathbf{Set} . \blacklozenge

Example 3.47. Let \mathcal{N} denote the category associated to the poset (\mathbb{N}, \leq) , and recall from Example 3.34 that we can identify a functor $F: \mathcal{N} \rightarrow \mathcal{N}$ with a non-decreasing sequence (F_0, F_1, F_2) of natural numbers, i.e. $F_0 \leq F_1 \leq F_2 \leq \dots$. If G is another functor, considered as a non-decreasing sequence, then what is a natural transformation $\alpha: F \rightarrow G$?

Since there is at most one morphism between two objects in a poset, each component $\alpha_n: F_n \rightarrow G_n$ has no “data”, it just tells us a fact: that $F_n \leq G_n$. And the naturality condition is vacuous: every square in a poset commutes. So a natural transformation between F and G exists iff $F_n \leq G_n$ for each n . Any two natural transformations $F \Rightarrow G$ are the same. In other words, $\mathcal{N}^{\mathcal{N}}$ is itself a poset; see Section 3.2.3. \blacklozenge

Exercise 3.48. Let \mathcal{C} be an arbitrary category and let \mathcal{P} be a poset, thought of as a category. Consider the following statements:

1. For any two functors $F, G: \mathcal{C} \rightarrow \mathcal{P}$, there is at most one natural transformation $F \rightarrow G$.
2. For any two functors $F, G: \mathcal{P} \rightarrow \mathcal{C}$, there is at most one natural transformation $F \rightarrow G$.

For each, if it is true, say why; if it is false, give a counterexample. \blacklozenge

Remark 3.49. Recall that in Remark 2.48 we said the category of posets is equivalent to the category of **Bool**-categories. We can now state the precise meaning of this sentence. First, there exists a category **Pos** in which the objects are posets and the morphisms are monotone maps. Second, there exists a category **Bool-Cat** in which the objects are **Bool**-categories and the morphisms are **Bool**-functors. We call these two categories equivalent because there exist functors $F: \mathbf{Pos} \rightarrow \mathbf{Bool-Cat}$ and $G: \mathbf{Bool-Cat} \rightarrow \mathbf{Pos}$ such that there exist natural isomorphisms $F.G \cong \text{id}_{\mathbf{Pos}}$ and $G.F \cong \text{id}_{\mathbf{Bool-Cat}}$ in the sense of Definition 3.40.

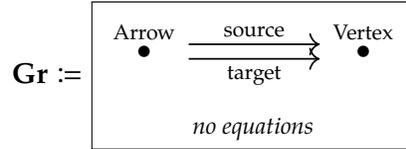
3.3.5 The category of instances on a schema

Definition 3.50. Suppose that \mathcal{C} is a database schema and $I, J: \mathcal{C} \rightarrow \mathbf{Set}$ are database instances. An *instance homomorphism* between them is a natural transformation $\alpha: I \rightarrow J$. Write $\mathcal{C}\text{-Inst} := \mathbf{Set}^{\mathcal{C}}$ to denote the functor category as defined in Definition 3.44.

We saw in Example 3.43 that $\mathbf{1-Inst}$ is equivalent to the category \mathbf{Set} . In this subsection, we will show that there is a schema whose instances are graphs and whose instance homomorphisms are graph homomorphisms.

Extended example: the category of graphs as a functor category You may find yourself back in the primordial ooze (first discussed in Section 2.3.2), because we have been using graphs to present categories; now we obtain graphs themselves as database

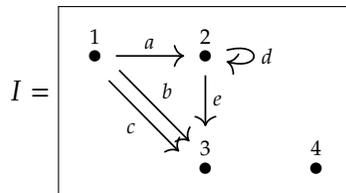
instances on a specific schema (which is itself a graph):



Here's an example \mathbf{Gr} -instance, i.e. set-valued functor $I: \mathbf{Gr} \rightarrow \mathbf{Set}$, in table form:

Arrow	source	target	Vertex
a	1	2	1
b	1	3	2
c	1	3	3
d	2	2	4
e	2	3	

Here $I(\text{Arrow}) = \{a, b, c, d, e\}$, and $I(\text{Vertex}) = \{1, 2, 3, 4\}$. One can draw the instance I as a graph:



Every row in the Vertex table is drawn as a vertex, and every row in the Arrow table is drawn as an arrow, connecting its specified source and target. Every possible graph can be written as a database instance on the schema \mathbf{Gr} , and every possible \mathbf{Gr} -instance can be represented as a graph.

Exercise 3.51. In Eq. (3.2), a graph is shown (forget the distinction between white and black nodes). Write down the corresponding \mathbf{Gr} -instance. (Do not be concerned that you are in the primordial ooze.) \diamond

Thus the objects in the category $\mathbf{Gr-Inst}$ are the graphs. In fact, the morphisms in $\mathbf{Gr-Inst}$ are called graph homomorphisms. Let's unwind this. Suppose that $G, H: \mathbf{Gr} \rightarrow \mathbf{Set}$ are functors (i.e. \mathbf{Gr} -instances); that is, they are objects $G, H \in \mathbf{Gr-Inst}$. A morphism $G \rightarrow H$ is a natural transformation $\alpha: G \rightarrow H$ between them; what does that entail?

By Definition 3.40, since \mathbf{Gr} has two objects, α consists of two components,

$$\alpha_{\text{Vertex}}: G(\text{Vertex}) \rightarrow H(\text{Vertex}) \quad \text{and} \quad \alpha_{\text{Arrow}}: G(\text{Arrow}) \rightarrow H(\text{Arrow})$$

both of which are morphisms in \mathbf{Set} . In other words, α consists of a function from vertices of G to vertices of H and a function from arrows of G to arrows of H . For these functions to constitute a graph homomorphism, they must "respect source and target" in the precise sense that the naturality condition, Eq. (3.9) holds. That is, for every

morphism in **Gr**, namely source and target, the following diagrams must commute:

$$\begin{array}{ccc}
 G(\text{Arrow}) & \xrightarrow{\alpha_{\text{Arrow}}} & H(\text{Arrow}) \\
 G(\text{source}) \downarrow & & \downarrow H(\text{source}) \\
 G(\text{Vertex}) & \xrightarrow{\alpha_{\text{Vertex}}} & H(\text{Vertex})
 \end{array}
 \qquad
 \begin{array}{ccc}
 G(\text{Arrow}) & \xrightarrow{\alpha_{\text{Arrow}}} & H(\text{Arrow}) \\
 G(\text{target}) \downarrow & & \downarrow H(\text{target}) \\
 G(\text{Vertex}) & \xrightarrow{\alpha_{\text{Vertex}}} & H(\text{Vertex})
 \end{array}$$

These may look complicated, but they say exactly what we want. We want the functions α_{Vertex} and α_{Arrow} to respect source and targets in G and H . The left diagram says “start with an arrow in G . You can either apply α to the arrow and then take its source in H , or you can take its source in G and then apply α to that vertex; either way you get the same answer”. The right-hand diagram says the same thing about targets.

Example 3.52. Consider the graphs G and H shown below



Here they are, written as database instances—i.e. set-valued functors—on **Gr**:

$G :=$	<table style="border-collapse: collapse; margin: 0 auto;"> <tr><th style="padding: 0 10px;">Arrow</th></tr> <tr><td style="padding: 0 10px;">a</td></tr> <tr><td style="padding: 0 10px;">b</td></tr> </table>	Arrow	a	b	<table style="border-collapse: collapse; margin: 0 auto;"> <tr><th style="padding: 0 10px;">source</th></tr> <tr><td style="padding: 0 10px;">1</td></tr> <tr><td style="padding: 0 10px;">2</td></tr> </table>	source	1	2	<table style="border-collapse: collapse; margin: 0 auto;"> <tr><th style="padding: 0 10px;">target</th></tr> <tr><td style="padding: 0 10px;">2</td></tr> <tr><td style="padding: 0 10px;">3</td></tr> </table>	target	2	3	<table style="border-collapse: collapse; margin: 0 auto;"> <tr><th style="padding: 0 10px;">Vertex</th></tr> <tr><td style="padding: 0 10px;">1</td></tr> <tr><td style="padding: 0 10px;">2</td></tr> <tr><td style="padding: 0 10px;">3</td></tr> </table>	Vertex	1	2	3		
Arrow																			
a																			
b																			
source																			
1																			
2																			
target																			
2																			
3																			
Vertex																			
1																			
2																			
3																			
$H :=$	<table style="border-collapse: collapse; margin: 0 auto;"> <tr><th style="padding: 0 10px;">Arrow</th></tr> <tr><td style="padding: 0 10px;">c</td></tr> <tr><td style="padding: 0 10px;">d</td></tr> <tr><td style="padding: 0 10px;">e</td></tr> </table>	Arrow	c	d	e	<table style="border-collapse: collapse; margin: 0 auto;"> <tr><th style="padding: 0 10px;">source</th></tr> <tr><td style="padding: 0 10px;">4</td></tr> <tr><td style="padding: 0 10px;">4</td></tr> <tr><td style="padding: 0 10px;">5</td></tr> </table>	source	4	4	5	<table style="border-collapse: collapse; margin: 0 auto;"> <tr><th style="padding: 0 10px;">target</th></tr> <tr><td style="padding: 0 10px;">5</td></tr> <tr><td style="padding: 0 10px;">5</td></tr> <tr><td style="padding: 0 10px;">5</td></tr> </table>	target	5	5	5	<table style="border-collapse: collapse; margin: 0 auto;"> <tr><th style="padding: 0 10px;">Vertex</th></tr> <tr><td style="padding: 0 10px;">4</td></tr> <tr><td style="padding: 0 10px;">5</td></tr> </table>	Vertex	4	5
Arrow																			
c																			
d																			
e																			
source																			
4																			
4																			
5																			
target																			
5																			
5																			
5																			
Vertex																			
4																			
5																			

The top row is G and the bottom row is H . They are offset so you can more easily complete the following exercise. ♦

Exercise 3.53. We claim that there is exactly one graph homomorphism $\alpha: G \rightarrow H$ such that $\alpha_{\text{Arrow}}(a) = d$.

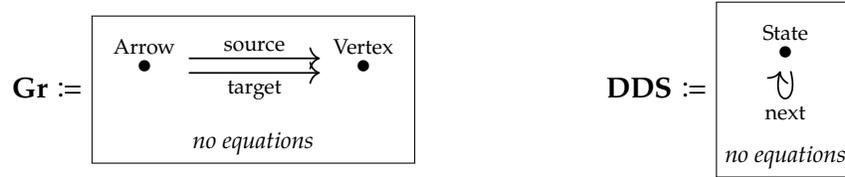
1. What is the other value of α_{Arrow} , and what are the three values of α_{Vertex} ?
2. Draw α_{Arrow} as two lines connecting the cells in the ID column of $G(\text{Arrow})$ to those in the ID column of $H(\text{Arrow})$. Similarly, draw α_{Vertex} as connecting lines.
3. Check the source column and target column and make sure that the matches are natural, i.e. that “alpha-then-source equals source-then-alpha” and similarly for “target”. ♦

3.4 Adjunctions and data migration

We have talked about how set-valued functors on a schema can be understood as filling that schema with data. But there are also functors between schemas. When the two sorts of functors are composed, data is migrated. This is the simplest form of data migration; more complex ways to migrate data come from using adjoints. All of this is the subject of this final section.

3.4.1 Pulling back data along a functor

To begin, we will migrate data between the graph-indexing schema \mathbf{Gr} and the loop schema, which we call \mathbf{DDS} , shown below



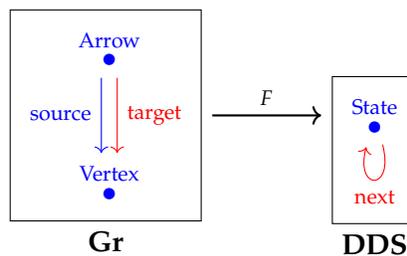
We begin by writing down a sample instance $I: \mathbf{DDS} \rightarrow \mathbf{Set}$ on this schema:

State	next
1	4
2	4
3	5
4	5
5	5
6	7
7	6

(3.10)

We call the schema \mathbf{DDS} to stand for discrete dynamical system. Once we migrate the data in Eq. (3.10) to data on \mathbf{Gr} , it will be a graph and we will draw it. In the meantime, think of the data in Eq. (3.10) like the states of a deterministic machine: at every point in time it is in a state and in the next instant it transforms to a next state.

We will use a functor connecting schemas in order to move data between them. The reader can create any functor she likes, but we will use a specific functor $F: \mathbf{Gr} \rightarrow \mathbf{DDS}$ to migrate data in a way that makes sense to us, the authors. Here we draw F , using colors to hopefully aid understanding:



F sends both objects of \mathbf{Gr} to the 'State' object of \mathbf{DDS} ; it sends the 'source' morphism to the identity morphism on 'State', and the 'target' morphism to the morphism 'next'.

Sample data on \mathbf{DDS} was given in Eq. (3.10); we think of it as a functor $I: \mathbf{DDS} \rightarrow \mathbf{Set}$. So now we have two functors as follows:

$$\mathbf{Gr} \xrightarrow{F} \mathbf{DDS} \xrightarrow{I} \mathbf{Set}.$$

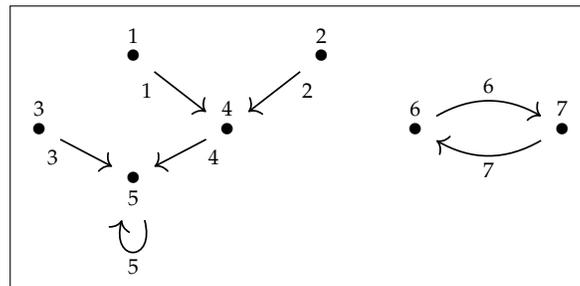
Objects in \mathbf{Gr} are sent by F to objects in \mathbf{DDS} , which are sent by I to objects in \mathbf{Set} , which are sets. Morphisms in \mathbf{Gr} are sent by F to morphisms in \mathbf{DDS} , which are

sent by I to morphisms in **Set**, which are functions. This defines a composite functor $F.I: \mathbf{Gr} \rightarrow \mathbf{Set}$. Both F and I respect identities and composition, so $F.I$ does too. Thus we have obtained an instance on **Gr**, i.e. we have converted our discrete dynamical system from Eq. (3.10) into a graph! What graph is it?

For an instance on **Gr**, we need to fill an Arrow table and a Vertex table. Both of these are sent by F to State, so let's fill both with the rows of State in Eq. (3.10). Similarly, since F sends 'source' to the identity and sends 'target' to 'next', we obtain the following tables:

Arrow	source	target	Vertex
1	1	4	1
2	2	4	2
3	3	5	3
4	4	5	4
5	5	5	5
6	6	7	6
7	7	6	7

Now that we have a graph, we can draw it.



Each arrow is labeled by its source vertex, as if to say "I am only what I do."

Exercise 3.54. Consider the functor $G: \mathbf{Gr} \rightarrow \mathbf{DDS}$ given by sending 'source' to 'next' and sending 'target' to the identity on 'State'. Migrate the same data, Eq. (3.10), using G . Write down the tables and draw the corresponding graph. \diamond

We call the above procedure, "pulling back data along a functor". We pulled back I along F .

Definition 3.55. Let C and D be categories and let $F: C \rightarrow D$ be a functor. For any set-valued functor $I: D \rightarrow \mathbf{Set}$, we refer to the functor $F.I: C \rightarrow \mathbf{Set}$ as the *pullback of I along F* .

Given a natural transformation $\alpha: I \rightarrow J$, there is a natural transformation $\alpha_F: F.I \rightarrow F.J$, whose component $(F.I)(c) \rightarrow (F.J)(c)$ for any $c \in \text{Ob}(C)$ is given by $(\alpha_F)_c := \alpha_{Fc}$.

Thus we have used F to define a functor which we denote $\Delta_F: D\text{-Inst} \rightarrow C\text{-Inst}$.

3.4.2 Adjunctions

In Section 1.5 we discussed Galois connections. These are adjunctions between posets. Now that we've defined categories and functors, we can discuss adjunctions in general. The relevance to databases is that the data migration functor Δ from Definition 3.55 always has two adjoints of its own: a left adjoint which we denote Σ and a right adjoint which we denote Π .

Recall that an adjunction between posets P and Q is a pair of monotone maps $f: P \rightarrow Q$ and $g: Q \rightarrow P$ that are *almost* inverses: we have

$$f(p) \leq q \text{ if and only if } p \leq g(q). \quad (3.11)$$

Recall from Section 3.2.3 that in a poset P , a hom-set $P(a, b)$ has one element when $a \leq b$, and no elements otherwise. Thus we can thus rephrase Eq. (3.11) as an isomorphism of sets $Q(f(p), q) \cong P(p, g(q))$: either both are one-element sets or both are 0-element sets. This suggests how to define adjunctions in the general case.

Definition 3.56. Let \mathcal{C} and \mathcal{D} be categories, and $L: \mathcal{C} \rightarrow \mathcal{D}$ and $R: \mathcal{D} \rightarrow \mathcal{C}$ be functors. We say that L is *left adjoint to* R (and that R is *right adjoint to* L) if, for any $c \in \mathcal{C}$ and $d \in \mathcal{D}$, there is an isomorphism of hom-sets

$$\alpha_{c,d}: \mathcal{C}(c, R(d)) \xrightarrow{\cong} \mathcal{D}(L(c), d)$$

that is natural in c and d .

To denote an adjunction we write $L \dashv R$, or in diagrams,

$$\begin{array}{ccc} & L & \\ \mathcal{C} & \begin{array}{c} \xrightarrow{\quad} \\ \perp \\ \xleftarrow{\quad} \end{array} & \mathcal{D} \\ & R & \end{array}$$

with the \perp -sign resting against the right adjoint.

Example 3.57. Recall that every poset \mathcal{P} can be regarded as a category. Galois connections between posets and adjunctions between the corresponding categories are exactly the same thing. \blacklozenge

Example 3.58. Let $B \in \text{Ob}(\mathbf{Set})$ be any set. There is an adjunction called “currying B ”, after the logician Haskell Curry:

$$\mathbf{Set} \begin{array}{c} \xrightarrow{-\times B} \\ \perp \\ \xleftarrow{(-)^B} \end{array} \mathbf{Set} \qquad \mathbf{Set}(A \times B, C) \cong \mathbf{Set}(A, C^B)$$

Abstractly we write it as on the left, but what this means is that for any sets A, C , there is a natural isomorphism as on the right.

To explain this, we need to talk about exponential objects in \mathbf{Set} . Suppose that B and C are sets. Then the set of functions $B \rightarrow C$ is also a set; let's denote it C^B . It's

written this way because if C has 10 elements and B has 3 elements then C^B has 10^3 elements, and more generally for any two finite sets $|C^B| = |C|^{|B|}$.

The idea of currying is that given sets A, B , and C , there is a one-to-one correspondence between functions $(A \times B) \rightarrow C$ and functions $A \rightarrow C^B$. Intuitively, if I have a function f of two variables a, b , I can “put off” entering the second variable: if you give me just a , I’ll return a function $B \rightarrow C$ that’s waiting for the B input. This is the curried version of f . ♦

Exercise 3.59. In Example 3.58, we discussed an adjunction between functors $- \times B$ and $(-)^B$. But we only said how these functors worked on objects: for any set X , they return sets $X \times B$ and X^B respectively.

1. Given a morphism $f: X \rightarrow Y$, what morphism should $- \times B: X \times B \rightarrow Y \times B$ return?
2. Given a morphism $f: X \rightarrow Y$, what morphism should $(-)^B: X^B \rightarrow Y^B$ return?
3. Consider the function $+: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, which sends $(a, b) \mapsto a + b$. Currying $+$, we get a certain function $p: \mathbb{N} \rightarrow \mathbb{N}^{\mathbb{N}}$. What is $p(3)$? ♦

Example 3.60. If you know some abstract algebra or topology, here are some other examples of adjunctions.

1. Free constructions: given any set you get a free group, free monoid, free ring, free vector space, etc.; each of these is a left adjoint. The corresponding right adjoint takes a group, a monoid, a ring, a vector space etc. and forgets the algebraic structure to return the “underlying set”.
2. Similarly, given a graph you get a free category or a free poset, as we discussed in Section 3.2.3; each is a left adjoint. The corresponding right adjoint is the “underlying graph” of a category or of a poset.
3. Discrete things: given any set you get a discrete graph, a discrete metric space, a discrete topological space; each of these is a left adjoint. The corresponding right adjoint is again “underlying set.”
4. Given a group, you can quotient by its commutator subgroup to get an abelian group; this is a left adjoint. The right adjoint is the inclusion of abelian groups into groups. ♦

3.4.3 Left and right pushforward functors, Σ and Π

Given $F: C \rightarrow D$, the data migration functor Δ_F turns D -instances into C -instances. This functor has both a left and a right adjoint:

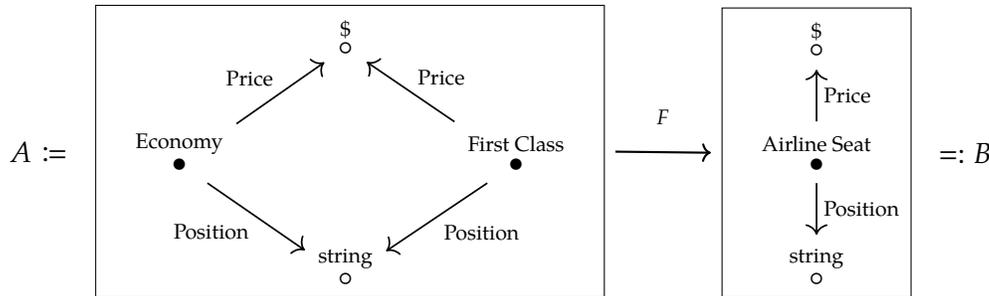
$$\begin{array}{ccc}
 & \xrightarrow{\Sigma_F} & \\
 C\text{-Inst} & \xleftarrow{\Delta_F} & D\text{-Inst} \\
 & \xrightarrow{\Pi_F} &
 \end{array}$$

Using the names Σ and Π in this context is fairly standard in category theory. In the case of databases, they have the following helpful mnemonic:

Migration Functor	Pronounced	Reminiscent of	Database idea
Δ	Delta	Duplicate or destroy	Duplicate or destroy tables or columns
Σ	Sigma	Sum	Union (sum up) data
Π	Pi	Product	Pair ⁶ and query data

Just like we used Δ_F to pull back any discrete dynamical system along $F: \mathbf{Gr} \rightarrow \mathbf{DDS}$ and get a graph, the migration functors Σ_F and Π_F can be used to turn any graph into a discrete dynamical system. Indeed, given instance $J: \mathbf{Gr} \rightarrow \mathbf{Set}$, we would get instances $\Sigma_F(J)$ and $\Pi_F(J)$ on \mathbf{DDS} . This, however, is quite technical, and we leave it to the adventurous reader to compute an example, with help perhaps from [Spi14], which explores the definitions of Σ and Π in detail. A less technical shortcut is simply to code up the computation in the open-source FQL software.

To get the basic idea across without getting mired in technical idea, here we shall instead discuss a very simple example. Recall the schemas from Eq. (3.4). We can set up a functor between them, the one sending black dots to black dots and white dots to white dots:



With this functor F in hand, we can transform any B -instance into an A -instance using Δ_F . Whereas Δ was interesting in the case of turning discrete dynamical systems into graphs in Section 3.4.1, it is not very interesting in this case. Indeed, it will just copy— Δ for duplicate—the rows in Airline seat into both Economy and First Class.

Δ_F has two adjoints, Σ_F and Π_F , both of which transform any A -instance I into a B -instance. The functor Σ_F does the what one would most expect by reading the names on each object: it will put into Airline Seat the union of Economy and First Class:

$$\Sigma_F(I)(\text{Airline Seat}) = I(\text{Economy}) \sqcup I(\text{First Class}).$$

The functor Π_F puts into Airline Seat the set of those pairs (e, f) where e is an Economy seat, f is a First Class seat, and e and f have the same price and position.

⁶This is more commonly called “join” by database programmers.

In this particular example, one imagine that there should be no such seats in a valid instance I , in which case $\Pi_F(I)(\text{Airline Seat})$ would be empty. But in other uses of these same schemas, Π_F can be a useful operation. For example, in the schema A replace the label ‘Economy’ by ‘Rewards Program’, and in B replace ‘Airline Seat’ by ‘First Class Seats’. Then the operation Π_F finds those first class seats that are also rewards program seats. This operation is a kind of database query; querying is the operation that databases are built for.

The moral is that complex data migrations can be specified by constructing functors F between schemas and using the “induced” functors Δ_F , Σ_F , and Π_F . Indeed, in practice essentially all useful migrations can be built in this way. Hence the language of categories provides a framework for specifying and reasoning about data migrations.

3.4.4 Single set summaries of databases

To give a stronger idea of the flavor of Σ and Π , we consider another special case, namely where the target category \mathcal{D} is equal to $\mathbf{1}$; see Exercise 3.5. In this case, there is exactly one functor $C \rightarrow \mathbf{1}$ for any C ; let’s denote it $! : C \rightarrow \mathbf{1}$.

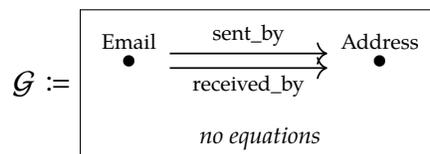
Exercise 3.61. Describe this functor $! : C \rightarrow \mathbf{1}$. Where does it send each object? What about each morphism? \diamond

We want to consider the data migration functors $\Sigma_! : C\text{-Inst} \rightarrow \mathbf{1}\text{-Inst}$ and $\Pi_! : C\text{-Inst} \rightarrow \mathbf{1}\text{-Inst}$. In Example 3.43, we saw that an instance on $\mathbf{1}$ is the same thing as a set. So let’s identify $\mathbf{1}\text{-Inst}$ with **Set**, and hence discuss

$$\Sigma_! : C\text{-Inst} \rightarrow \mathbf{Set} \quad \text{and} \quad \Pi_! : C\text{-Inst} \rightarrow \mathbf{Set}.$$

Given any schema C and instance $I : C \rightarrow \mathbf{Set}$, we will get sets $\Sigma_!(I)$ and $\Pi_!(I)$. Thinking of these sets as database instances, each corresponds to a single one-column table—a controlled vocabulary—summarizing an entire database instance on the schema C .

Consider the following schema



Here’s a sample instance $I : \mathcal{G} \rightarrow \mathbf{Set}$:

Email	sent_by	received_by	Address
Em_1	Bob	Grace	Bob
Em_2	Grace	Pat	Doug
Em_3	Bob	Emory	Emory
Em_4	Sue	Doug	Grace
Em_5	Doug	Sue	Pat
Em_6	Bob	Bob	Sue

Exercise 3.62. Note that \mathcal{G} is isomorphic to the schema \mathbf{Gr} . In Section 3.3.5 we saw that instances on \mathbf{Gr} are graphs. Draw the above instance I as a graph. \diamond

Now we have a unique functor $! : \mathcal{G} \rightarrow \mathbf{1}$, and we want to say what $\Sigma_!(I)$ and $\Pi_!(I)$ give us as single-set summaries. First, $\Sigma_!(I)$ tells us all the emailing groups—the “connected components”—in I :

$$\frac{\mathbf{1}}{\text{Bob-Grace-Pat-Emory} \\ \text{Sue-Doug}} \quad \Bigg|$$

This form of summary, involving identifying entries into common groups, or quotients, is typical of Σ -operations.

The functor $\Pi_!(I)$ lists the emails from I which were sent from a person to her- or him-self.

$$\frac{\mathbf{1}}{\text{Em}_6} \quad \Bigg|$$

This is again a sort of query, selecting the entries that fit the criterion of self-to-self emails. Again, this is typical of Π -operations.

Where do these facts—that $\Pi_!$ and $\Sigma_!$ act the way we said—come from? Everything follows from the definition of adjoint functors (3.56): indeed we hope this, together with the examples given in Example 3.60, give the reader some idea of how general and useful adjunctions are, both in mathematics and in database theory.

One more point: while we will not spell out the details, we note that these operations are also examples of constructions known as small colimits and limits in \mathbf{Set} . We end this chapter by exploring these key category theoretic constructions. The reader should keep in mind that, in general and not just for functors to $\mathbf{1}$, Σ -operations are built from colimits in \mathbf{Set} , and Π -operations are built from limits in \mathbf{Set} .

3.5 Bonus: An introduction to limits and colimits

What do products of sets, the results of $\Pi_!$ -operations on database instances, and meets in a poset all have in common? The answer, as we shall see, is that they are all examples of limits. Similarly, disjoint unions of sets, the results of $\Sigma_!$ -operations on database instances, and joins in a poset are all colimits. Let’s begin with limits.

Recall that $\Pi_!$ takes a database instance $I : C \rightarrow \mathbf{Set}$ and turns it into a set $\Pi_!(I)$. More generally, limits turn a functor $F : C \rightarrow \mathcal{D}$ into an object of \mathcal{D} .

3.5.1 Terminal objects and products

Terminal objects and products are each a sort of limit. Let’s discuss them in turn.

Terminal objects The most basic limit is a terminal object.

Definition 3.63. Let C be a category. Then an object Z in C is a *terminal object* if, for each object C of C , there exists a unique morphism $!: C \rightarrow Z$.

Since this unique morphism exists for all objects in C , we say that terminal objects have a *universal property*.

Example 3.64. In **Set**, any set with exactly one element is a terminal object. Why? Consider some such set $\{\bullet\}$. Then for any other set C we need to check that there is exactly one function $!: C \rightarrow \{\bullet\}$. This unique function is the one that does the only thing that can be done: it maps each element $c \in C$ to the element $\bullet \in \{\bullet\}$. \blacklozenge

Exercise 3.65. Let (P, \leq) be a poset. Show that $z \in P$ is a terminal object if and only if it is maximal: that is, if and only if for all $c \in P$ we have $c \leq z$. \blacklozenge

Exercise 3.66. What is the terminal object in the category **Cat**? (Hint: recall Exercise 3.61.) \blacklozenge

Exercise 3.67. Not every category has a terminal object. Find one that doesn't. \blacklozenge

Proposition 3.68. *All terminal objects are isomorphic.*

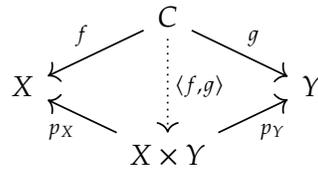
Proof. This is a simple, but powerful standard argument. Suppose Z and Z' are both terminal objects in some category C . Then there exist unique maps $a: Z \rightarrow Z'$ and $b: Z' \rightarrow Z$. Composing these, we get maps $a.b: Z \rightarrow Z$ and $b.a: Z' \rightarrow Z'$. But we know there is a map $\text{id}_Z: Z \rightarrow Z$, and since Z is terminal there is only one such map. So we must have $a.b = \text{id}_Z$. Similarly, we find that $b.a = \text{id}_{Z'}$. Thus a (and its inverse b) is an isomorphism. \square

Remark 3.69 (“The limit” vs. “a limit”). Not only are all terminal objects isomorphic, there is a unique isomorphism between any two. We hence say “terminal objects are unique up to unique isomorphism.” To a category theorist, this is very nearly the same thing as saying “all terminal objects are equal”. Thus we often abuse terminology and talk of ‘the’ terminal object, rather than ‘a’ terminal object. We will do the same for any sort of limit or colimit, e.g. speak of “the product” of two sets, rather than “a product”.

Products Products are slightly more complicated to formalize than terminal objects are, but they are familiar in practice.

Definition 3.70. Let C be a category, and let X, Y be objects in C . A *product* of X and Y is an object, denoted $X \times Y$, together with morphisms $p_X: X \times Y \rightarrow X$ and $p_Y: X \times Y \rightarrow Y$ such that for all objects C together with morphisms $f: C \rightarrow X$ and $g: C \rightarrow Y$, there exists a unique morphism $C \rightarrow X \times Y$, denoted $\langle f, g \rangle$, for which the following diagram

commutes:



We will try to bring this down to earth in Example 3.71. Before we do, note that $X \times Y$ is an object equipped with morphisms to X and Y . Roughly speaking, it is like “the best object-equipped-with-morphisms-to- X -and- Y ” in all of C , in the sense that any other object-equipped-with-morphisms-to- X -and- Y maps to it uniquely. This is called a *universal property*. It’s customary to use a dotted line to indicate the unique morphism that exists because of some universal property.

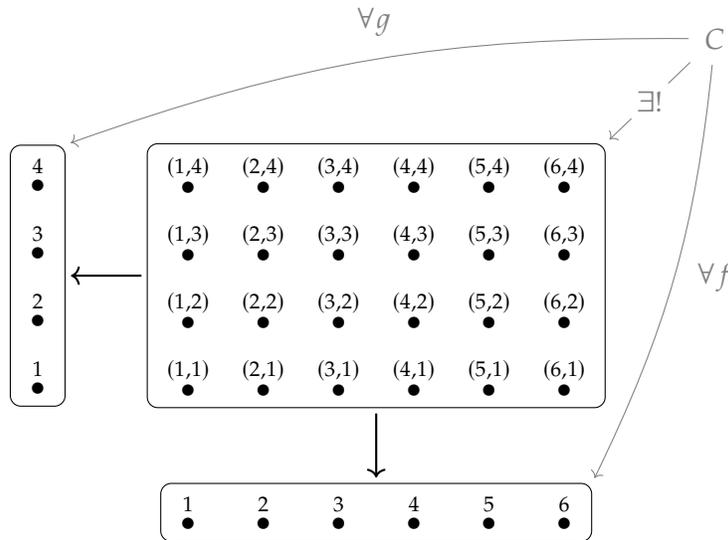
Example 3.71. In **Set**, a product of two sets X and Y is their usual cartesian product

$$X \times Y := \{(x, y) \mid x \in X, y \in Y\},$$

which comes with two projections $p_X: X \times Y \rightarrow X$ and $p_Y: X \times Y \rightarrow Y$, given by $p_X(x, y) = x$ and $p_Y(x, y) = y$.

Given any set C with functions $f: C \rightarrow X$ and $g: C \rightarrow Y$, the unique map from C to $X \times Y$ such that the required diagram commutes is given by $\langle f, g \rangle(c) := (f(c), g(c))$.

Here is a picture of the product of sets 4 and 6.



Exercise 3.72. Let (P, \leq) be a poset, and suppose that $x, y \in P$. Show that their product and their meet are equivalent, $(x \times y) \cong (x \wedge y)$. ♦

Example 3.73. Given two categories C and D , their product $C \times D$ may be given as follows. The objects of this category are pairs (c, d) , where c is an object of C and d is an object of D . Similarly, morphisms $(c, d) \rightarrow (c', d')$ are pairs (f, g) where $f: c \rightarrow c'$ is a morphism in C and $g: d \rightarrow d'$ is a morphism in D . Composition of morphisms is simply given by composing each entry in the pair separately, so $(f, g) \cdot (f', g') = (f \cdot f', g \cdot g')$. ♦

- Exercise 3.74.*
1. What are the identity morphisms in a product category $\mathcal{C} \times \mathcal{D}$?
 2. Why is composition in a product category associative?
 3. What is the product category $\mathbf{1} \times \mathbf{2}$? ◇
 4. What is the product category $P \times Q$ when P and Q are posets?

These two constructions, terminal objects and products, are subsumed by the notion of limit.

3.5.2 Limits

We'll get a little abstract. Consider the definition of product. This says that given any pair of maps $X \xleftarrow{f} C \xrightarrow{g} Y$, there exists a unique map $C \rightarrow X \times Y$ such that certain diagrams commute. This has the flavor of being terminal—there is a unique map to $X \times Y$ —but it's a bit more complicated. How are the two ideas related?

It turns out that products *are* terminal objects, but of a different category, which we'll call $\mathbf{Cone}(X, Y)$, *the category of cones over X and Y in \mathcal{C}* . It will turn out—see Exercise 3.75—that $X \xleftarrow{p_X} X \times Y \xrightarrow{p_Y} Y$ is a terminal object in $\mathbf{Cone}(X, Y)$.

An object of $\mathbf{Cone}(X, Y)$ is simply a pair of maps $X \xleftarrow{f} C \xrightarrow{g} Y$. A morphism from $X \xleftarrow{f} C \xrightarrow{g} Y$ to $X \xleftarrow{f'} C' \xrightarrow{g'} Y$ in $\mathbf{Cone}(X, Y)$ is a morphism $a: C \rightarrow C'$ in \mathcal{C} such that the following diagram commutes:

$$\begin{array}{ccccc}
 & & C & & \\
 & f & \swarrow & & \searrow g \\
 X & & & & Y \\
 & f' & \swarrow & & \searrow g' \\
 & & C' & &
 \end{array}$$

Exercise 3.75. Check that a product $X \xleftarrow{p_X} X \times Y \xrightarrow{p_Y} Y$ is exactly the same as a terminal object in $\mathbf{Cone}(X, Y)$. ◇

We're now ready for the abstract definition. Don't worry if the details are unclear; the main point is that it is possible to unify terminal objects, maximal elements, meets, products of sets, posets, categories, and many other familiar friends. In fact, they're all just terminal objects in different categories.

Recall from Definition 3.41 that formally speaking, a diagram in \mathcal{C} is just a functor $D: \mathcal{J} \rightarrow \mathcal{C}$. Here \mathcal{J} is called the i

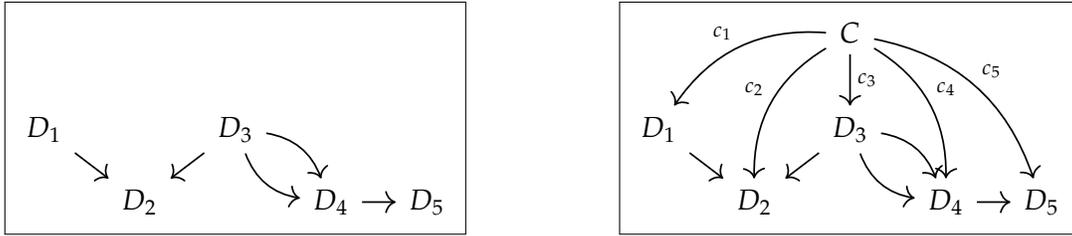
Definition 3.76. Let $D: \mathcal{J} \rightarrow \mathcal{C}$ be a diagram. A *cone* (C, c_*) over D is

- (i) an object $C \in \mathcal{C}$, called the *cone point*.
- (ii) for each object $j \in \mathcal{J}$, a morphism $c_j: C \rightarrow D(j)$, called the j^{th} *structure map* obeying
 - (a) for each $f: i \rightarrow j$ in \mathcal{J} , we have $c_i = c_j \circ D(f)$.

A *morphism of cones* $(C, c_*) \rightarrow (C', c'_*)$ is a morphism $a: C \rightarrow C'$ in \mathcal{C} such that for all $j \in \mathcal{J}$ we have $c_j = a \circ c'_j$. Cones and their morphisms form a category $\mathbf{Cone}(D)$.

The *limit* of D is the terminal object in the category of $\mathbf{Cone}(D)$. Its cone point is often called the *limit object* and its structure maps are often called *projection maps*.

For visualization purposes, if $D: \mathcal{J} \rightarrow \mathcal{C}$ looks like the diagram to the left, then a cone on it shown in the diagram to the right:



Here, any two parallel paths that start at C are considered the same.

Example 3.77. Terminal objects are limits where the indexing category is empty, $\mathcal{J} = \emptyset$. ♦

Example 3.78. Products are limits where the indexing category consists of two objects v, w and no arrows, $\mathcal{J} = \begin{bmatrix} v & w \\ \bullet & \bullet \end{bmatrix}$. ♦

3.5.3 Finite limits in \mathbf{Set}

Recall that this discussion was inspired by wanting to understand $\Pi_!$ operations, and in particular $\Pi_!$. We can now see that a database instance $I: \mathcal{C} \rightarrow \mathbf{Set}$ is a diagram in \mathbf{Set} . The functor $\Pi_!$ takes the limit of this diagram. In this subsection we give a formula describing the result. This captures *all finite limits in \mathbf{Set}* .

In database theory, we work with categories \mathcal{C} that are presented by a finite graph. We won't explain the details, but it's in fact enough just to work with this graph: as far as limits are concerned, the equations in \mathcal{C} don't matter. For consistency with the rest of this section, let's denote the database schema by \mathcal{J} instead of \mathcal{C} .

Theorem 3.79. *Let \mathcal{J} be a category presented by the finite graph (V, A, s, t) together with some equations, and let $D: \mathcal{J} \rightarrow \mathbf{Set}$ be a set-valued functor. Write $V = \{v_1, \dots, v_n\}$. The set*

$$\lim_{\mathcal{J}} D := \{(d_1, \dots, d_n) \mid d_i \in D(v_i) \text{ and for all } a: v_i \rightarrow v_j \in A, \text{ we have } D(a)(d_i) = d_j\},$$

together with the projection maps $p_i: (\lim_{\mathcal{J}} D) \rightarrow D(v_i)$ given by $p_i(d_1, \dots, d_n) := d_i$, is a limit of D .

Example 3.80. If J is the empty graph \square , then $n = 0$: there are no vertices. There is exactly one empty tuple $()$, which vacuously satisfies the properties, so we've constructed the limit as the singleton set $\{()\}$ consisting of just the empty tuple. The limit of the empty diagram, i.e. the terminal object in \mathbf{Set} is the singleton set. See Remark 3.69. ♦

Exercise 3.81. Show that the limit formula in Theorem 3.79 works for products. See Example 3.78. ♦

Exercise 3.82. If $D: \mathbf{1} \rightarrow \mathbf{Set}$ is a functor, what is the limit of D ? Compute it using Theorem 3.79, and check your answer against Definition 3.76. \diamond

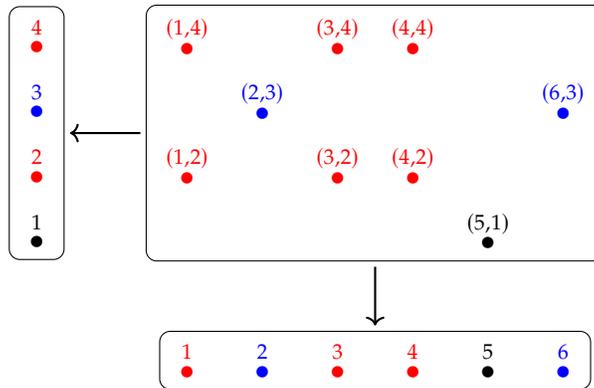
Pullbacks In particular, the condition that the limit of $D: \mathcal{J} \rightarrow \mathbf{Set}$ selects tuples (d_1, \dots, d_n) such that $D(a)(d_i) = d_j$ for each morphism $a: i \rightarrow j$ in \mathcal{J} allows us to use limits to select data that satisfies certain equations or constraints. This is what allows us to express queries in terms of limits. Here is an example.

Example 3.83. If J is presented by the *cospan* graph $\begin{array}{ccc} x & \xrightarrow{f} & a \\ & & \xleftarrow{g} & y \end{array}$, then its limit is known as a *pullback*. Given the diagram $X \xrightarrow{f} A \xleftarrow{g} Y$, the pullback is the cone shown on the left below:



The fact that the diagram commutes means that the diagonal arrow c_a is in some sense superfluous, so one generally denotes pullbacks by dropping the diagonal arrow, naming the cone point $X \times_A Y$, and adding the \lrcorner symbol, as shown to the right above.

Here is a picture to help us unpack the definition in \mathbf{Set} . We take $X = \underline{4}$, $Y = \underline{6}$, and A to be the set of colors {red, blue, black}.



The functions $f: \underline{4} \rightarrow A$ and $g: \underline{6} \rightarrow A$ are expressed in the coloring of the dots: for example, $f(2) = f(4) = \text{red}$, while $g(5) = \text{black}$. The pullback selects pairs $(i, j) \in \underline{4} \times \underline{6}$ such that $f(i)$ and $g(j)$ have the same color. \diamond

3.5.4 A brief note on colimits

Just like upper bounds have as a dual concept, namely that of lower bounds, so limits have a dual concept: colimits. To expose the reader to this concept, we provide a succinct definition of these using opposite categories and opposite categories. The point, however, is just exposure; we will return to explore colimits in detail in Chapter 6.

Exercise 3.84. Recall from Definition 3.19 that every category C has an opposite C^{op} . Let $F: C \rightarrow D$ be a functor. How should we define its opposite, $F^{\text{op}}: C^{\text{op}} \rightarrow D^{\text{op}}$? That is, how should F^{op} do on objects, and how should it act on morphisms? \diamond

Definition 3.85. Given a category C we say that a *cocone* in C is a cone in C^{op} .

Given a diagram $D: \mathcal{J} \rightarrow C$, we may take the limit of the functor $D^{\text{op}}: \mathcal{J}^{\text{op}} \rightarrow C^{\text{op}}$. This is a cone in C^{op} , and so by definition a cocone in C . The *colimit* of D is this cocone.

Definition 3.85 is like a compressed file: useful for transmitting quickly, but completely useless for working with. In order to work with it, we will need to unpack it; we do so later in Chapter 6 when we discuss electric circuits.

3.6 Summary and further reading

Congratulations on making it through the longest chapter in the book! We apologize for the length, but this chapter had a lot of work to do. Namely it introduced the “big three” of category theory—categories, functors, and natural transformations—as well as discussing adjunctions, limits, and very briefly colimits.

That’s really quite a bit of material. For more on all these subjects, one can consult any standard book on category theory, of which there are many. The bible (old, important, seminal, and requires a priest to explain it) is [Mac98]; another thorough introduction is [Bor94]; a logical perspective is given in [Awo10]; a computer science perspective is given in [BW90]; math students should probably read [Lei14] or [Rie17]; a general audience might start with [Spi14].

We presented categories from a database perspective, because data is pretty ubiquitous in our world. A database schema—i.e. a system of interlocking tables—can be captured by a category C , and filling it with data corresponds to a functor $C \rightarrow \mathbf{Set}$. Here \mathbf{Set} is the category of sets, perhaps the most important category to mathematicians.

The perspective of using category theory to model databases has been rediscovered several times. It seems to have first been discussed by various authors around the mid-90’s [IP94; CD95; PS95; TG96]. Bob Rosebrugh and collaborators took it much further in a series of papers including [FGR03; JR02; RW92]. Most of these authors tend to focus on sketches, which are more expressive categories. Spivak rediscovered the idea again quite a bit later, but focused on categories rather than sketches, so as to have all three data migration functors Δ, Σ, Π ; see [Spi12; SW15b]. The version of this story presented in the chapter, including the white and black nodes in schemas, is part of a larger theory of algebraic databases, where a programming language such as Java is attached to a database. This is worked out in [Sch+17], and its use in database integration projects can be found in [SW15a; Wis+15].

Before we leave this chapter, we want to emphasize two things: coherence conditions and universal constructions.

Coherence conditions In the definitions of category, functor, and natural transformations, we always had one or more structures that satisfied one or more conditions: for categories it was about associativity and unitality of composition, for functors it was about respecting composition and identities, and for natural transformations it was the naturality condition. These conditions are often called *coherence conditions*: we want the various structures to cohere, to work well together, rather than to flop around unattached.

Understanding why these particular structures and coherence conditions are “the right ones” is more science than mathematics: we empirically observe that certain combinations result in ideas that are both widely applicable and also strongly compositional. That is, we become satisfied with coherence conditions when they result in beautiful mathematics down the road.

Universal constructions Universal constructions are one of the most important themes of category theory. Roughly speaking, one gives some specified shape in a category and says “find me the best solution!” And category theory comes back and says “do you want me to approximate from the left or the right (colimit or limit)?” You respond, and either there is a best solution or there is not. If there is, it’s called the (co)limit; if there’s not we say “the (co)limit does not exist”.

Even data migration fits this form. We say “find me the closest thing in \mathcal{D} that matches my \mathcal{C} -instance using my functor $F: \mathcal{C} \rightarrow \mathcal{D}$.” In fact this approach—known as Kan extensions—in fact subsumes the others. One of the two founders of category theory, Saunders Mac Lane, has a section in his book [Mac98] called “All concepts are Kan extensions”, a big statement, no?

Collaborative design: Profunctors, categorification, and monoidal categories

4.1 Can we build it?

When designing a large-scale system, many different fields of expertise are joined into a single project. Thus the whole project team is divided into multiple sub-teams, each of which is working on a sub-project. And we recurse downward: the sub-project is again factored into sub-sub-projects, each with their own team. One could refer to this sort of hierarchical design process as *collaborative design*, or co-design. In this chapter, we discuss a mathematical theory of co-design, due to Andrea Censi [Cen15].

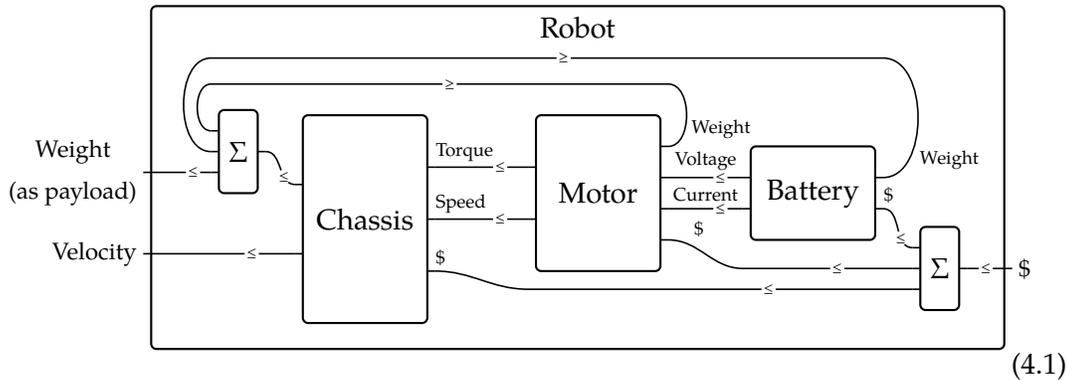
Consider just one level of this hierarchy: a project and a set of teams working on it. Each team is supposed to *provide* resources—sometimes called “functionalities”—to the project, but the team also *requires* resources in order to do so. Different design teams must be allowed to plan and work independently from one another in order for progress to be made. Yet the design decisions made by one group effect the design decisions others can make: if A wants more space in order to provide a better radio speaker, then B must use less space. So these teams—though ostensibly working independently—are dependent on each other after all.

The combination of dependence and independence is crucial for progress to be made, and yet it can cause major problems. When a team requires more resources than it originally expected to require, or if it cannot provide the resources that it originally claimed it could provide, the usual response is for the team to issue a design-change notice. But these effect neighboring teams: if team A now requires more than originally claimed, team B may have to change their design. Thus these design-change notices can ripple through the system through feedback loops and cause whole projects to fail [S+15].

As an example, consider the design problem of creating a robot to carry some load at some velocity. The top-level planner breaks the problem into three design teams: team chassis, team motor, and team battery. Each of these teams could break up into multiple parts and the process repeated, but let's remain at the top level and consider the resources produced and the resources required by each of our three teams.

The chassis in some sense provides all the functionality—it carries the load at the velocity—but it requires some things in order to do so. It requires money, of course, but more to the point it requires a source of torque and speed. These are supplied by the motor, which in turn needs voltage and current from the battery. Both the motor and the battery cost money, but more importantly they need to be carried by the chassis: they become part of the load. A feedback loop is created: the chassis must carry all the weight, even that of the parts that power the chassis. A heavier battery might provide more energy to power the chassis, but is the extra power worth the heavier load?

In the following picture, each part—chassis, motor, battery, and robot—is shown as a box with ports on the left and right. The functionalities, or resources produced by the part are on the left of the box, and the resources required by the part are on the right.



The boxes marked Σ correspond to summing inputs. These boxes are not to be designed, but we will see later that they fit easily into the same conceptual framework. Note also the \leq 's on each wire; they indicate that if box A requires a resource that box B produces, then A 's requirement must be less-than-or-equal-to B 's production.

To formalize this a bit more, let's call diagrams like the one above *co-design diagrams*. Each of the wires in a co-design diagram represents a poset of resources. For example, in Eq. (4.1) every wire corresponds to a resource type—weights, velocities, torques, speeds, costs, voltages, and currents—where resources of each type can be ordered from less useful to more useful. In general, these posets do not have to be linear orders, though in the above cases each will likely correspond to a linear order: $\$10 \leq \20 , $5W \leq 6W$, and so on.

Each of the boxes in a co-design diagram corresponds to what we call a *feasibility relation*. A feasibility relation matches resource production with requirements. For every pair $(p, r) \in P \times R$, where P is the poset of resources to be produced and R is the poset of resources to be required, the box says “true” or “false”—feasible or

infeasible—for that pair. In other words, “yes I can provide p given r ” or “no, I cannot provide p given r ”.

Feasibility relations hence define a function $\Phi: P \times R \rightarrow \mathbf{Bool}$. For a function $\Phi: P \times R \rightarrow \mathbf{Bool}$ to make sense as a feasibility relation, however, there are two conditions:

- (a) If $\Phi(p, r) = \mathbf{true}$ and $p' \leq p$, then $\Phi(p', r) = \mathbf{true}$.
- (b) If $\Phi(p, r) = \mathbf{true}$ and $r \leq r'$ then $\Phi(p, r') = \mathbf{true}$.

These conditions, which we will see again in Definition 4.1, say that if you can produce p given resources r , you can (a) also produce less $p' \leq p$ with the same resources r , and (b) also produce p given more resources $r' \geq r$. We will see that these two conditions are formalized by requiring Φ to be a monotone map $P^{\text{op}} \times R \rightarrow \mathbf{Bool}$.

A *co-design problem*, represented by a co-design diagram, asks us to find the composite of some feasibility relations. It asks, for example, given these capabilities of the chassis, motor, and battery teams, can we, together, build a robot? Indeed, a co-design diagram factors a problem—for example, that of designing a robot—into interconnected subproblems, as in Eq. (4.1). Once the feasibility relation is worked out for each of the subproblems, i.e. the inner boxes in the diagram, the mathematics provides an algorithm producing the feasibility relation of the whole outer box. This process can be recursed downward, from the largest problem to tiny subproblems.

In this chapter, we will understand co-design problems in terms of enriched profunctors, in particular **Bool**-profunctors. A **Bool**-profunctor is like a bridge connecting one poset to another. We will show how the co-design framework gives rise to a structure known as a compact closed category, and that any compact closed category can interpret the sorts of wiring diagrams we see in Eq. (4.1).

4.2 Enriched profunctors

In this section we will understand how co-design problems form a category. Along the way we will develop some abstract machinery that will allow us to replace poset design spaces with other enriched categories.

4.2.1 Feasibility relationships as **Bool**-profunctors

The theory of co-design is based on posets: each resource—e.g. velocity, torque, or \$—is structured as a poset. The order $x \leq y$ represents the *availability of x given y* , i.e. that whenever you have y , you also have x . For example, in our poset of wattage, if $5\text{W} \leq 10\text{W}$, it means that whenever we are provided 10W , we implicitly also have 5W .

We know from Section 2.3.2 that a poset X can be conceived of as a **Bool**-category. Given $x, y \in X$, we have $X(x, y) \in \mathbb{B}$; this value responds to the assertion “ x is available given y ,” marking it either **true** or **false**.

Our goal is to see feasibility relations as **Bool**-profunctors, which are a special case of something called enriched profunctors. Indeed, we hope that this chapter will give

you some intuition for profunctors, arising from the table

Bool -category	poset
Bool -functor	monotone map
Bool -profunctor	feasibility relation

Because enriched profunctors are a touch abstract, we first concretely discuss **Bool**-profunctors as feasibility relations. Recall that if $\mathcal{X} = (X, \leq_X)$ is a poset, then its opposite $\mathcal{X}^{\text{op}} = (X, \geq)$ has $x \geq y$ iff $y \leq x$.

Definition 4.1. Let $\mathcal{X} = (X, \leq_X)$ and $\mathcal{Y} = (Y, \leq_Y)$ be posets. A *feasibility relation* for \mathcal{X} given \mathcal{Y} is a monotone map

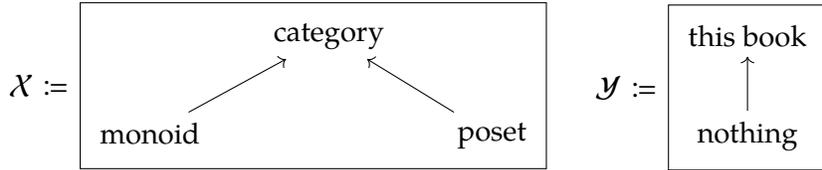
$$\Phi: \mathcal{X}^{\text{op}} \times \mathcal{Y} \rightarrow \mathbf{Bool} \tag{4.2}$$

We denote this by $\Phi: \mathcal{X} \dashv \mathcal{Y}$.

Given $x \in X$ and $y \in Y$, if $\Phi(x, y) = \text{true}$ we say *x can be obtained given y*.

As mentioned in the introduction, the requirement that Φ is monotone says that if $x' \leq_X x$ and $y \leq_Y y'$ then $\Phi(x, y) \leq_{\mathbf{Bool}} \Phi(x', y')$. In other words, if x can be obtained given y , and if x' is available given x , then x' can be obtained given y . And if furthermore y is available given y' , then x' can also be obtained given y' .

Exercise 4.2. Suppose we have the posets



1. Draw the Hasse diagram for the poset $\mathcal{X}^{\text{op}} \times \mathcal{Y}$.
2. Write down a profunctor $\Lambda: \mathcal{X} \dashv \mathcal{Y}$ and, reading $\Lambda(x, y) = \text{true}$ as “my uncle can explain x given y ”, give an interpretation of the fact that the preimage of true forms an upper set in $\mathcal{X}^{\text{op}} \times \mathcal{Y}$. ♦

To generalize the notion of feasibility relation, we must notice that the symmetric monoidal poset **Bool** has more structure than just that of a symmetric monoidal poset: as mentioned in Exercise 2.59, **Bool** is a quantale. That means it has all joins \vee , and a closure operation, which we’ll write $\Rightarrow: \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$. By definition, this operation satisfies the property that for all $b, c, d \in \mathbb{B}$ one has

$$b \wedge c \leq d \quad \text{iff} \quad b \leq (c \Rightarrow d). \tag{4.3}$$

The operation \Rightarrow is given by the following table:

c	d	$c \Rightarrow d$
true	true	true
true	false	false
false	true	true
false	false	true

(4.4)

Exercise 4.3. Show that \Rightarrow as defined in Eq. (4.4) indeed satisfies Eq. (4.3). \diamond

On an abstract level, it is the fact that **Bool** is a quantale which makes everything in this chapter work; any other (commutative) quantale also defines a way to interpret co-design diagrams. For example, we could use the quantale **Cost**, which would describe not *whether* x is available given y but the *cost* of obtaining x given y ; see Example 2.19 and Definition 2.28.

4.2.2 \mathcal{V} -profunctors

We are now ready to recast Eq. (4.2) in abstract terms. Recall the notions of enriched product (Definition 2.51), enriched functor (Definition 2.46), and commutative quantale (Definition 2.55).

Definition 4.4. Let $\mathcal{V} = (V, \leq, I, \otimes)$ be a commutative quantale, and let \mathcal{X} and \mathcal{Y} be \mathcal{V} -categories. A \mathcal{V} -profunctor from \mathcal{X} to \mathcal{Y} , denoted $\Phi: \mathcal{X} \rightarrow \mathcal{Y}$, is a \mathcal{V} -functor

$$\Phi: \mathcal{X}^{\text{op}} \times \mathcal{Y} \rightarrow \mathcal{V}.$$

Note that a \mathcal{V} -functor must have \mathcal{V} -categories for domain and codomain, so here we are considering \mathcal{V} as enriched in itself; see Remark 2.63.

Exercise 4.5. Show that a \mathcal{V} -profunctor (Definition 4.4) is the same as a function $\Phi: \text{Ob}(\mathcal{X}) \times \text{Ob}(\mathcal{Y}) \rightarrow V$ such that for any $x, x' \in \mathcal{X}$ and $y, y' \in \mathcal{Y}$ the following inequality holds in \mathcal{V} :

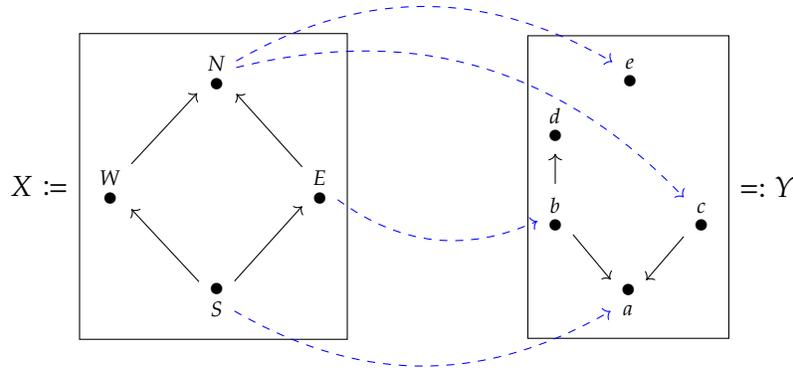
$$\mathcal{X}(x', x) \otimes \Phi(x, y) \otimes \mathcal{Y}(y, y') \leq \Phi(x', y'). \quad \diamond$$

Exercise 4.6. Is it true that a **Bool**-profunctor, as in Definition 4.4 is exactly the same as a feasibility relation, as in Definition 4.1, once you peel back all the jargon? Or is there some subtle difference? \diamond

We know that Definition 4.4 is quite abstract. But have no fear, we will take you through it in pictures.

Example 4.7 (Bool-profunctors and their interpretation as bridges). Let's discuss Definition 4.4 in the case $\mathcal{V} = \mathbf{Bool}$. One way to imagine a **Bool**-profunctor $\Phi: X \rightarrow Y$ is in terms of building bridges between two cities. Recall that a poset (a **Bool**-category) can be drawn using a Hasse diagram. We'll think of the poset as a city, and each vertex in it as some point of interest. An arrow $A \rightarrow B$ in the Hasse diagram means that there exists a way to get from point A to point B in the city. So what's a profunctor?

A profunctor is just a bunch of bridges connecting points in one city to points in another. Let's see a specific example. Here is a picture of a **Bool**-profunctor $\Phi: X \rightarrow Y$:



Both X and Y are posets, e.g. with $W \leq N$ and $b \leq a$. With bridges coming from the profunctor in blue, one can now use both paths within the cities and the bridges get from points in city X to points in city Y . For example, since there is a path from N to e and E to a , we have $\Phi(N, e) = \text{true}$ and $\Phi(E, a) = \text{true}$. On the other hand, since there is no path from W to d , we have $\Phi(W, d) = \text{false}$.

In fact, one could put a box around this entire picture and see a new poset with $W \leq N \leq c \leq a$, etc. This is called the *collage* of Φ ; we'll explore this in more detail in Section 4.3.3. ♦

Exercise 4.8. We can express Φ as a matrix where the (m, n) th entry is the value of $\Phi(m, n) \in \mathbb{B}$. Fill out the **Bool**-matrix:

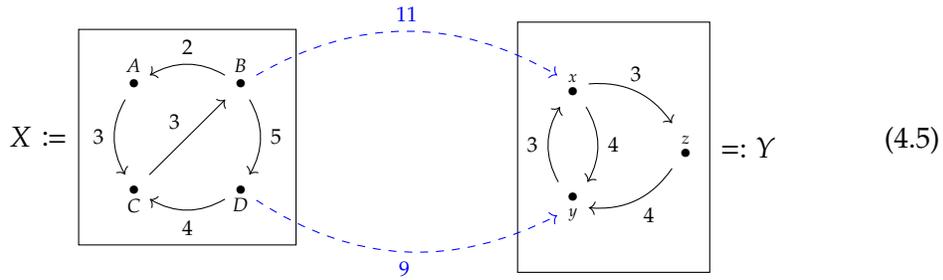
Φ	a	b	c	d	e
N	?	?	?	?	true
E	true	?	?	?	?
W	?	?	?	false	?
S	?	?	?	?	?

♦

We'll call this the *feasibility matrix* of Φ .

Example 4.9 (Cost-profunctors and their interpretation as bridges). Let's now consider **Cost**-profunctors. Again we can view these as bridges, but this time our bridges are labelled by their length. Recall from Definition 2.34 and Eq. (2.17) that **Cost**-categories are Lawvere metric spaces, and can be depicted using weighted graphs. We'll think of such a weighted graph as a chart of distances between points in a city, and generate a **Cost**-profunctor by building a few bridges between the cities.

Here is a depiction of a **Cost**-profunctor $\Phi: X \rightarrow Y$:



The distance from a point x in city X to a point y in city Y is given by the shortest path that runs from x through X , then across one of the bridges, and then through Y to the destination y . So for example

$$\Phi(B, x) = 11, \quad \Phi(A, z) = 20, \quad \Phi(C, y) = 17. \quad \blacklozenge$$

Exercise 4.10. Fill out the **Cost**-matrix:

Φ	x	y	z	
A	?	?	20	
B	11	?	?	\blacklozenge
C	?	17	?	
D	?	?	?	

Remark 4.11 (Computing profunctors via matrix multiplication). We can give an algorithm for computing the above distance matrix using matrix multiplication. First, just like in Eq. (2.19), we can begin with the labelled graphs in Eq. (4.5) and read off the matrices of arrow labels for X , Y , and Φ :

M_X	A	B	C	D	M_Φ	x	y	z	M_Y	x	y	z
A	0	∞	∞	∞	A	∞	∞	∞	x	0	4	3
B	2	0	∞	5	B	11	∞	∞	y	3	0	∞
C	∞	3	0	∞	C	∞	∞	∞	z	∞	4	0
D	∞	∞	4	0	D	∞	9	∞				

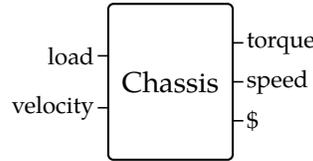
Recall from Section 2.5.3 that the matrix of distances d_Y for **Cost**-category Y can be obtained by taking the matrix power of M_Y with smallest entries, and similarly for X . The matrix of distances for the profunctor Φ will be equal to $d_X * M_\Phi * d_Y$. In fact, since X has four elements and Y has three, we also know that $\Phi = M_X^4 * M_\Phi * M_Y^3$.

Exercise 4.12. Calculate $M_X^4 * M_\Phi * M_Y^3$, remembering to do matrix multiplication according to the $(\min, +)$ -formula for matrix multiplication in the quantale **Cost**; see Eq. (2.25).

Your answer should agree with what you got in Exercise 4.10; does it? \blacklozenge

4.2.3 Back to co-design diagrams

Each box in a co-design diagram has a left-hand and a right-hand side, which in turn consist of a collection of ports, which in turn are labeled by posets. For example, consider the chassis box below:

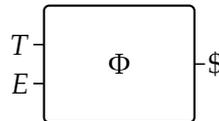


Its left side consists of two ports—one for load and one for velocity—and these are the functionality that the chassis produces. Its right side consists of three ports—one for torque, one for speed, and one for \$—and these are the resources that the chassis requires. Each of these resources is to be taken as a poset. For example, load might be the poset $([0, \infty], \leq)$, where an element $x \in [0, \infty]$ represents the idea “I can handle any load up to x .”, while \$ might be the two-element poset $\{\text{up_to_}\$100, \text{more_than_}\$100\}$, where the first element of this set is less than the second.

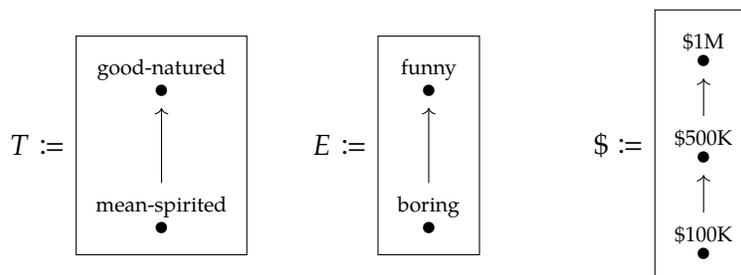
We then multiply—i.e. we take the product poset—of all posets on the left, and similarly for those on the right. The box then represents a feasibility relation between the results. For example, the chassis box above represents a feasibility relation

$$\text{Chassis: load} \times \text{velocity} \rightarrow \text{torque} \times \text{speed} \times \$$$

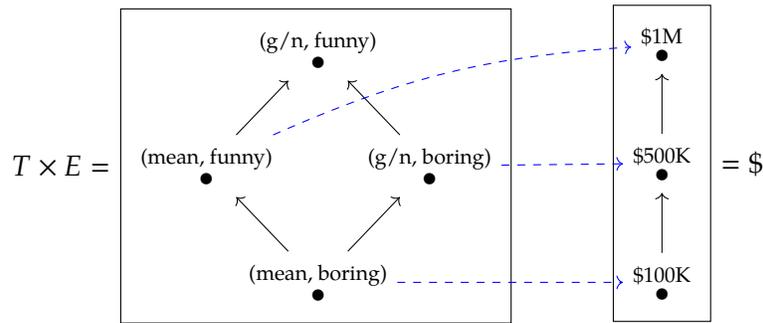
Let’s walk through this a bit more concretely. Consider the design problem of filming a movie, where you must pit the tone and entertainment value against the cost. A feasibility relation describing this situation details what tone and entertainment value can be obtained at each cost; as such, it is described by a feasibility relation $\Phi: (T \times E) \rightarrow \$$. We represent this by the box



where T , E , and $\$$ are the posets drawn below:



A possible feasibility relation is then described by the profunctor



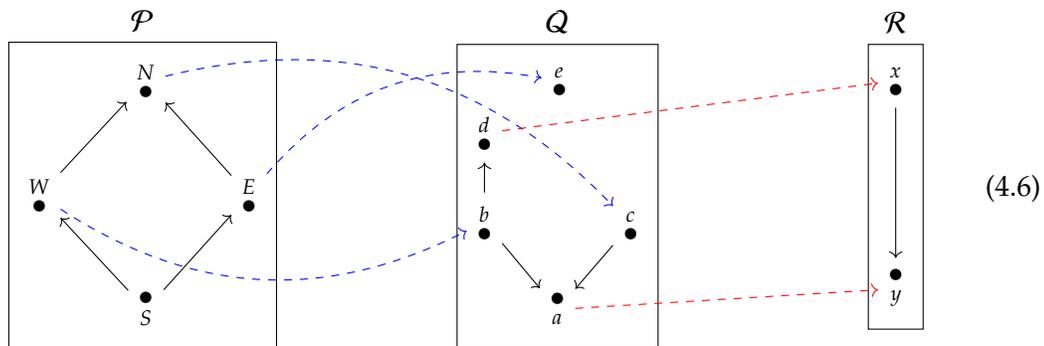
This says, for example, that a good-natured but boring movie costs \$500K to produce (of course, the producers would also be happy to get \$1M).

4.3 Categories of profunctors

There is a category **Feas** whose objects are posets and whose morphisms are feasibility relations. In order to describe it, we must give the composition formula and the identities, and prove that they satisfy the properties of being a category: unitality and associativity.

4.3.1 Composing profunctors

If feasibility relations are to be morphisms, we need to give a formula for composing two of them in series. Imagine you have cities \mathcal{P} , \mathcal{Q} , and \mathcal{R} and you have bridges—and hence feasibility matrices—connecting these cities, say $\Phi: \mathcal{P} \rightarrow \mathcal{Q}$ and $\Psi: \mathcal{Q} \rightarrow \mathcal{R}$.



The feasibility matrices for Φ (in blue) and Ψ (in red) are:

Φ	a	b	c	d	e
N	true	false	true	false	false
E	true	false	true	false	true
W	true	true	true	true	false
S	true	true	true	true	true

Ψ	x	y
a	false	true
b	true	true
c	false	true
d	true	true
e	false	false

As in Remark 2.66, we personify a quantale as a navigator. So imagine a navigator is trying to give a feasibility matrix $\Phi.\Psi$ for getting from \mathcal{P} to \mathcal{R} . How should this be done? Basically, for every pair $p \in \mathcal{P}$ and $r \in \mathcal{R}$, the navigator searches through \mathcal{Q} for a way-point q , somewhere both to which we can get from p AND from which we can get to r . It is true that we can navigate from p to r if any q can be a way-point; this is a big OR over all possible q . The composition formula is thus:

$$(\Phi.\Psi)(p, r) := \bigvee_{q \in \mathcal{Q}} \Phi(p, q) \wedge \Psi(q, r). \tag{4.7}$$

But as we have said, this can be thought of as matrix multiplication. In our example, the result is

$\Phi.\Psi$	x	y
N	false	true
E	false	true
W	true	true
S	true	true

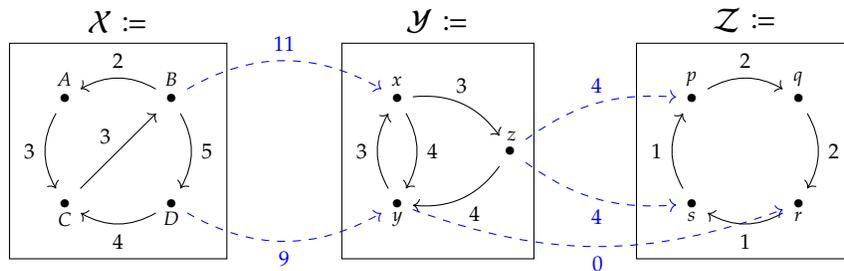
and one can check that this answers the question, “can you get from here to there” in Eq. (4.6): you can’t get from N to x but you can get from N to y .

The formula (4.7) is written in terms of the quantale **Bool**, but it works for arbitrary commutative quantales. We give the following definition.

Definition 4.13. Let \mathcal{V} be a commutative quantale, let \mathcal{X}, \mathcal{Y} , and \mathcal{Z} be \mathcal{V} -categories, and let $\Phi: \mathcal{X} \rightarrow \mathcal{Y}$ and $\Psi: \mathcal{Y} \rightarrow \mathcal{Z}$ be \mathcal{V} -profunctors. We define their *composite*, denoted $\Phi.\Psi: \mathcal{X} \rightarrow \mathcal{Z}$ as the map given by the formula

$$(\Phi.\Psi)(p, r) = \bigvee_{q \in \mathcal{Q}} (\Phi(p, q) \otimes \Psi(q, r)).$$

Exercise 4.14. Consider the **Cost**-profunctors $\Phi: \mathcal{X} \rightarrow \mathcal{Y}$ and $\Psi: \mathcal{Y} \rightarrow \mathcal{Z}$ shown below:



Fill in the matrix:

$\Phi.\Psi$	p	q	r	s
A	?	24	?	?
B	?	?	?	?
C	?	?	?	?
D	?	?	9	?

◇

4.3.2 The categories \mathcal{V} -Prof and Feas

A composition rule suggests a category, and there is indeed a category where the objects are **Bool**-categories and the morphisms are **Bool**-profunctors. To make this work more generally, however, we need to add one technical condition.

Recall that a poset is a skeletal poset if whenever $x \leq y$ and $y \leq x$, we have $x = y$. A skeletal poset is also known as a partially ordered set. We say a quantale is skeletal if its underlying poset is skeletal; **Bool** and **Cost** are skeletal quantales.

Theorem 4.15. *For any skeletal commutative quantale \mathcal{V} ,¹ there is a category $\mathbf{Prof}_{\mathcal{V}}$ whose objects are \mathcal{V} -categories \mathcal{X} , whose morphisms are \mathcal{V} -profunctors $\mathcal{X} \rightarrow \mathcal{Y}$, and with composition defined as in Definition 4.13.*

Definition 4.16. We define $\mathbf{Feas} := \mathbf{Prof}_{\mathbf{Bool}}$.

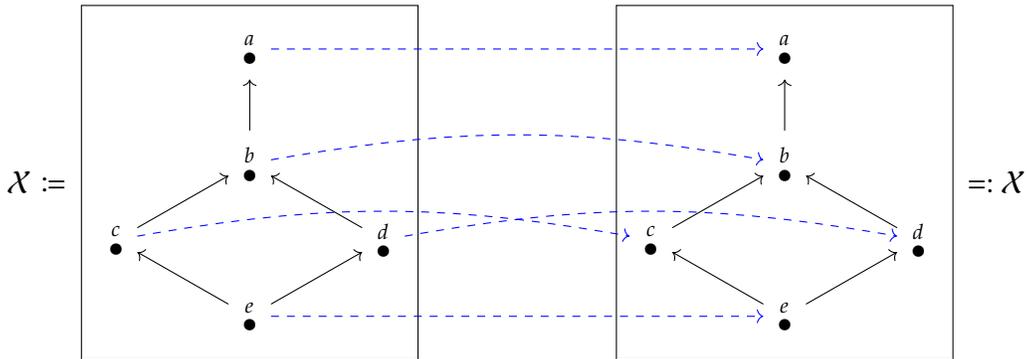
At this point perhaps you have two questions in mind. What are the identity morphisms? And why did we need to specialize to skeletal quantales? It turns out these two questions are closely related.

Define the *unit profunctor* $U_{\mathcal{X}}$ on a \mathcal{V} -category \mathcal{X} by the formula

$$U_{\mathcal{X}}(x, y) := \mathcal{X}(x, y). \tag{4.8}$$

How do we interpret this? Recall that, by Definition 2.28, \mathcal{X} already assigns to each pair of elements $x, y \in \mathcal{X}$ an hom-object $\mathcal{X}(x, y) \in \mathcal{V}$. The unit profunctor $U_{\mathcal{X}}$ just assigns each pair (x, y) that same object.

In the **Bool** case the unit profunctor on some poset \mathcal{X} can be drawn like this:



Obviously, composing a feasibility relation with with the unit leaves it unchanged, which is the content of Lemma 4.18.

Exercise 4.17. Choose a not-too-simple **Cost**-category \mathcal{X} . Give a bridge-style diagram for the unit profunctor $U_{\mathcal{X}} : \mathcal{X} \rightarrow \mathcal{X}$. ◇

Lemma 4.18. *Composing any profunctor $\Phi : \mathcal{P} \rightarrow \mathcal{Q}$ with either unit profunctor, $U_{\mathcal{P}}$ or $U_{\mathcal{Q}}$, returns Φ :*

$$U_{\mathcal{P}}.\Phi = \Phi = \Phi.U_{\mathcal{Q}}$$

¹From here on, as in Chapter 2, whenever we speak of quantale we mean commutative quantales.

Proof. We show that $U_{\mathcal{P}}.\Phi = \Phi$ holds; proving $\Phi = \Phi.U_Q$ is similar. Fix $p \in P$ and $q \in Q$. Since \mathcal{V} is skeletal, to prove the equality it's enough to show $\Phi \leq U_{\mathcal{P}}.\Phi$ and $U_{\mathcal{P}}.\Phi \leq \Phi$. We have one direction:

$$\Phi(p, q) = I \otimes \Phi(p, q) \leq \mathcal{P}(p, p) \otimes \Phi(p, q) \leq \bigvee_{p_1 \in P} (\mathcal{P}(p, p_1) \otimes \Phi(p_1, q)) = (U_{\mathcal{P}}.\Phi)(p, q). \tag{4.9}$$

For the other direction, we must show $\bigvee_{p_1 \in P} (\mathcal{P}(p, p_1) \otimes \Phi(p_1, q)) \leq \Phi(p, q)$. But by definition of join, this holds iff $\mathcal{P}(p, p_1) \otimes \Phi(p_1, q) \leq \Phi(p, q)$ is true for each $p_1 \in P$. This follows from Definitions 2.28 and 4.4:

$$\mathcal{P}(p, p_1) \otimes \Phi(p_1, q) = \mathcal{P}(p, p_1) \otimes \Phi(p_1, q) \otimes I \leq \mathcal{P}(p, p_1) \otimes \Phi(p_1, q) \otimes Q(q, q) \leq \Phi(p, q). \quad \square$$

- Exercise 4.19.* 1. Justify each of the four steps ($=, \leq, \leq, =$) in Eq. (4.9).
 2. In the case $\mathcal{V} = \mathbf{Bool}$, we can directly show each of the four steps in Eq. (4.9) is actually an equality. How? \diamond

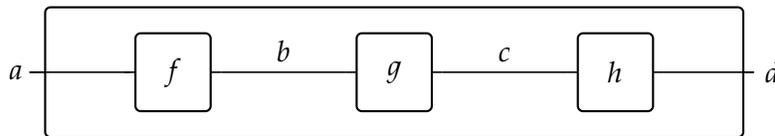
Composition of profunctors is also associative; we leave the proof to you.

Lemma 4.20. *Serial composition of profunctors is associative: given profunctors $\Phi: \mathcal{P} \rightarrow \mathcal{Q}$, $\Psi: \mathcal{Q} \rightarrow \mathcal{R}$, and $\Upsilon: \mathcal{R} \rightarrow \mathcal{S}$, we have*

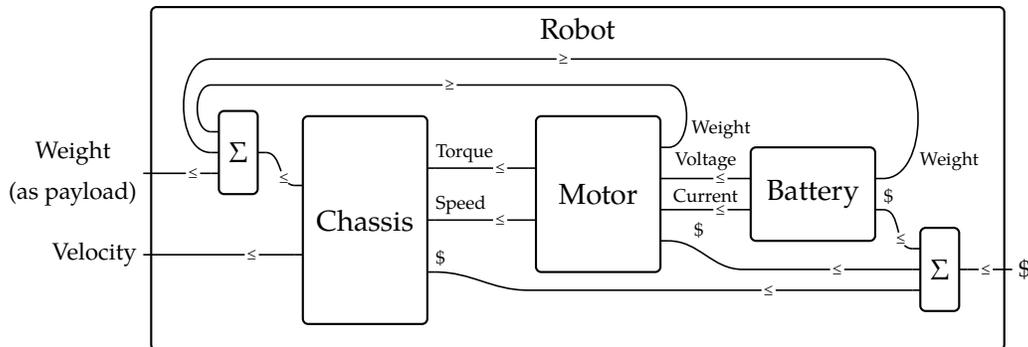
$$(\Phi.\Psi).\Upsilon = \Phi.(\Psi.\Upsilon).$$

Exercise 4.21. Prove Lemma 4.20. (Hint: remember to use the fact that \mathcal{V} is skeletal.) \diamond

So, feasibility relations form a category. Since this is the case, we can describe feasibility relations using string diagrams for categories. Recall, however, string diagrams for categories are very simple. Indeed, each box can only have one input and one output, and they're connected in a line:



On the other hand, we have seen that feasibility relations are the building blocks of co-design problems, and we know that co-design problems can be depicted in a much richer way, for example:



This hints that the category **Feas** has more structure. We saw wiring diagrams where boxes can have multiple inputs and outputs in Chapter 2; there they depicted morphisms in a monoidal poset. On other hand the boxes in the wiring diagrams of Chapter 2 could not have labels, like the boxes in a co-design problem, and similarly, we know that **Feas** is a proper category, not just a poset. To understand these diagrams then, we will have to introduce a new structure. This will be a *categorified* monoidal poset; these are known, not surprisingly, as *monoidal categories*.

Remark 4.22. While we have chosen to define **Prof** $_{\mathcal{V}}$ only for skeletal (commutative) quantales in Theorem 4.15, it is not too hard to work with non-skeletal ones. There are two straightforward ways to do this. First, we might let the morphisms of **Prof** $_{\mathcal{V}}$ be isomorphism classes of \mathcal{V} -profunctors. This is analogous to the trick we will use when defining the category **Cospan** $_C$ in Definition 6.32. Second, we might relax what we mean by category, only requiring composition to be unital and associative ‘up to isomorphism’. This is also a type of categorification.

In the next section we’ll discuss categorification and introduce monoidal categories. First though, we finish this section by discussing why profunctors are called profunctors, and by formally introducing the notation for profunctors called a *collage*.

4.3.3 Fun profunctor facts: companions, conjoins, collages

Companions and conjoins

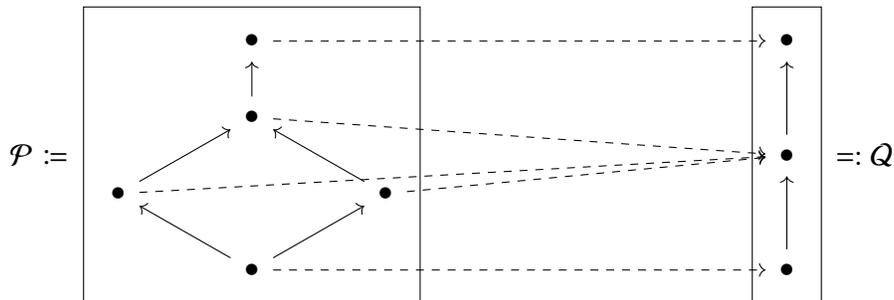
Recall that a poset is a **Bool**-category and a monotone map is a **Bool**-functor. We said above that a profunctor is a generalization of a functor; how so?

In fact, every \mathcal{V} -functor gives rise to two \mathcal{V} -profunctors, called the companion and the conjoint.

Definition 4.23. Let $F: \mathcal{P} \rightarrow \mathcal{Q}$ be a \mathcal{V} -functor. The *companion* of F , denoted $\widehat{F}: \mathcal{P} \rightarrow \mathcal{Q}$ and the *conjoint* of F , denoted $\check{F}: \mathcal{Q} \rightarrow \mathcal{P}$ are defined to be the following \mathcal{V} -profunctors:

$$\widehat{F}(p, q) := \mathcal{Q}(F(p), q) \quad \text{and} \quad \check{F}(q, p) := \mathcal{Q}(q, F(p))$$

Let’s consider the **Bool** case again. One can think of a monotone map $F: \mathcal{P} \rightarrow \mathcal{Q}$ as a bunch of arrows, one coming out of each vertex $p \in P$ and landing at some vertex $F(p) \in Q$.



This looks like the pictures of bridges connecting cities, and if one regards the above picture in that light, they are seeing the companion \widehat{F} . But now mentally reverse every dotted arrow, and the result would be bridges Q to P . This is a profunctor $Q \rightarrow P$! We call it \check{F} .

Example 4.24. For any poset \mathcal{P} , there is an identity functor $\text{id}: \mathcal{P} \rightarrow \mathcal{P}$. Its companion and conjoint agree $\widehat{\text{id}} = \check{\text{id}}: \mathcal{P} \rightarrow \mathcal{P}$. The resulting profunctor is in fact the unit profunctor, $U_{\mathcal{P}}$ as defined in Eq. (4.8). \blacklozenge

Exercise 4.25. Explain why the companion $\widehat{\text{id}}$ of $\text{id}: \mathcal{P} \rightarrow \mathcal{P}$ really has the formula given in Eq. (4.8). \blacklozenge

Example 4.26. Consider the function $+: \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$, sending a triple (a, b, c) of real numbers to $a + b + c \in \mathbb{R}$. This function is monotonic, because if $(a, b, c) \leq (a', b', c')$ —i.e. if $a \leq a'$ and $b \leq b'$, and $c \leq c'$ —then obviously $a + b + c \leq a' + b' + c'$. Thus it has a companion and a conjoint.

Its companion $\widehat{+}: (\mathbb{R} \times \mathbb{R} \times \mathbb{R}) \rightarrow \mathbb{R}$ is the function that sends (a, b, c, d) to true if $a + b + c \leq d$ and to false otherwise. \blacklozenge

Exercise 4.27. Let $+: \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ be as in Example 4.26. What is its conjoint $\check{+}$? \blacklozenge

Remark 4.28 (\mathcal{V} -Adjoints). Recall from Definition 1.70 the definition of Galois connection between posets \mathcal{P} and \mathcal{Q} . The definition of adjoint can be extended from the **Bool**-enriched setting (of posets and monotone maps) to the \mathcal{V} -enriched setting for arbitrary monoidal posets \mathcal{V} . In that case, the definition of a \mathcal{V} -adjunction is a pair of \mathcal{V} -functors $F: \mathcal{P} \rightarrow \mathcal{Q}$ and $G: \mathcal{Q} \rightarrow \mathcal{P}$ such that the following holds for all $p \in \mathcal{P}$ and $q \in \mathcal{Q}$.

$$\mathcal{P}(p, G(q)) \cong \mathcal{Q}(F(p), q) \quad (4.10)$$

Exercise 4.29. Let \mathcal{V} be a skeletal quantale, let \mathcal{P} and \mathcal{Q} be \mathcal{V} -categories, and let $F: \mathcal{P} \rightarrow \mathcal{Q}$ and $G: \mathcal{Q} \rightarrow \mathcal{P}$ be \mathcal{V} -functors.

1. Show that F and G are \mathcal{V} -adjoints (as in Eq. (4.10)) if and only if the companion of the former equals the conjoint of the latter: $\widehat{F} = \check{G}$.
2. Use this to prove that $\widehat{\text{id}} = \check{\text{id}}$, as was stated in Example 4.24. \blacklozenge

Collage of a profunctor

We have been drawing profunctors as bridges connecting cities. One may get an inkling that given a \mathcal{V} -profunctor $\Phi: X \rightarrow Y$ between \mathcal{V} -categories \mathcal{X} and \mathcal{Y} , we have turned Φ into a some sort of new \mathcal{V} -category that has \mathcal{X} on the left and \mathcal{Y} on the right. This works for any \mathcal{V} and profunctor Φ , and is called the collage construction.

Definition 4.30. Let \mathcal{V} be a quantale, let \mathcal{X} and \mathcal{Y} be \mathcal{V} -categories, and let $\Phi: \mathcal{X} \rightarrow \mathcal{Y}$ be a \mathcal{V} -profunctor. The *collage of Φ* , denoted $\mathbf{Col}(\Phi)$ is the \mathcal{V} -category defined as follows:

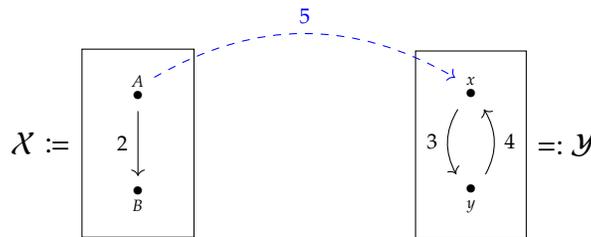
1. $\text{Ob}(\mathbf{Col}(\Phi)) := \text{Ob}(\mathcal{X}) \sqcup \text{Ob}(\mathcal{Y})$;

2. For any $a, b \in \text{Ob}(\mathbf{Col}(\Phi))$, define $\mathbf{Col}(\Phi)(a, b) \in \mathcal{V}$ to be

$$\mathbf{Col}(\Phi)(a, b) := \begin{cases} \mathcal{X}(a, b) & \text{if } a, b \in \mathcal{X} \\ \Phi(a, b) & \text{if } a \in \mathcal{X}, b \in \mathcal{Y} \\ \emptyset & \text{if } a \in \mathcal{Y}, b \in \mathcal{X} \\ \mathcal{Y}(a, b) & \text{if } a, b \in \mathcal{Y} \end{cases}$$

There are obvious functors $i_{\mathcal{X}}: \mathcal{X} \rightarrow \mathbf{Col}(\Phi)$ and $i_{\mathcal{Y}}: \mathcal{Y} \rightarrow \mathbf{Col}(\Phi)$, sending each object and morphism to “itself”, called *collage inclusions*.

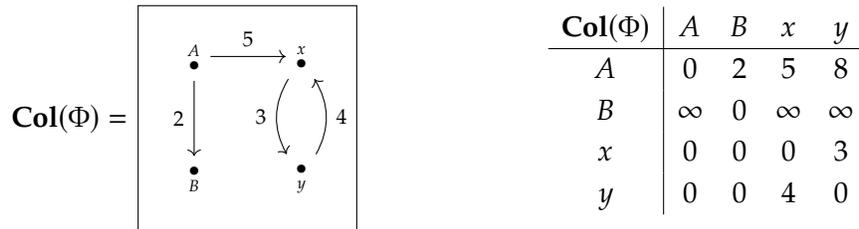
Example 4.31. For example, consider the following picture of a **Cost**-profunctor $\Phi: \mathcal{X} \rightarrow \mathcal{Y}$:



It corresponds to the following matrices

\mathcal{X}	A	B	Φ	x	y	\mathcal{Y}	x	y
A	0	2	A	5	8	x	0	3
B	∞	0	B	∞	∞	y	4	0

A generalized Hasse diagram of the collage can be obtained by simply taking the union of the Hasse diagrams for \mathcal{X} and \mathcal{Y} , and adding in the bridges as arrows. Given the above profunctor Φ , we draw the Hasse diagram for $\mathbf{Col}(\Phi)$ below left, and the **Cost**-matrix representation of the resulting **Cost**-category on the right:



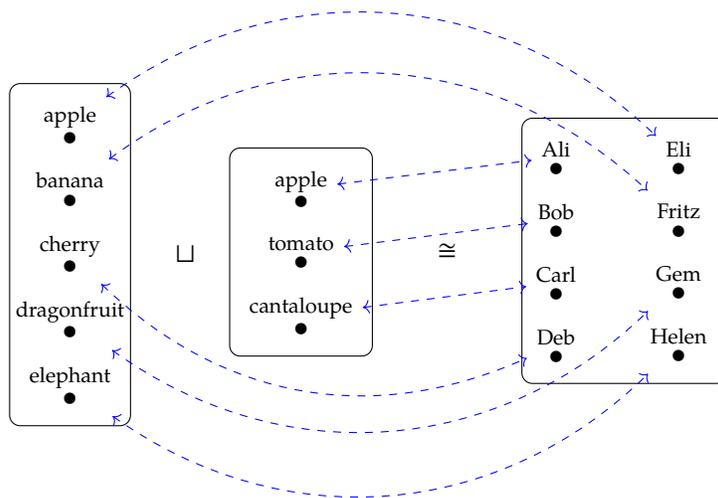
4.4 Categorification

Here we switch gears, to discuss a general concept called *categorification*. We will begin again with the basics, categorifying several of the notions we’ve encountered already. The goal is to define compact closed categories and their feedback-style wiring diagrams. At that point we will return to the story of co-design, and \mathcal{V} -profunctors in general, and show that they do in fact form a compact closed category, and thus interpret the diagrams we’ve been drawing since Eq. (4.1).

4.4.1 The basic idea of categorification

The general idea of categorification is that we take a thing we know and add structure to it, so that what were formerly *properties* become *structures*. We do this in such a way that we can recover the thing we categorified by forgetting this new structure. This is rather vague; let's give an example.

Basic arithmetic concerns properties of the natural numbers \mathbb{N} , such as the fact that $5 + 3 = 8$. One way to categorify \mathbb{N} is to use the category **FinSet** of finite sets and functions. To obtain a categorification, we replace the brute 5, 3, and 8 with sets of that many elements, say $\bar{5} = \{\text{apple, banana, cherry, dragonfruit, elephant}\}$, $\bar{3} = \{\text{apple, tomato, cantaloupe}\}$, and $\bar{8} = \{\text{Ali, Bob, Carl, Deb, Eli, Fritz, Gem, Helen}\}$ respectively. We also replace $+$ with disjoint union of sets \sqcup , and the brute property of equality with the structure of an isomorphism. What makes this a good categorification is that, having made these replacements, the analogue of $5 + 3 = 8$ is still true: $\bar{5} \sqcup \bar{3} \cong \bar{8}$.



In this categorified world, we have more structure available to talk about the relationships between objects, so we can be more precise about how they relate to each other. Thus it's not the case that $\bar{5} \sqcup \bar{3}$ is *equal* to our chosen eight-element set $\bar{8}$, but more precisely that there exists an invertible function comparing the two, showing that they are isomorphic in the *category* **FinSet**.

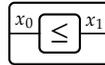
Note that in the above construction we made a number of choices; here we must beware. Choosing a good categorification, like designing a good algebraic structure like that of posets or quantales, is part of the art of mathematics. There is no prescribed way to categorify, and the success of a chosen categorification is often empirical: its richer structure allows us more insights into the subject we want to model.

As another example, an empirically pleasing way to categorify posets is to categorify them as, well, categories. In this case, rather than the brute property "there exists a morphism $a \rightarrow b$ ", denoted $a \leq b$ or $\mathcal{P}(a, b) = \text{true}$, we instead say "here is a set of

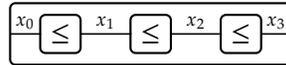
morphisms $a \rightarrow b''$. We get a hom-set rather than a hom-Boolean. In fact—to state this in a way straight out of the primordial ooze—just as posets are **Bool**-categories, ordinary categories are actually **Set**-categories.

4.4.2 A reflection on wiring diagrams

Suppose we have a poset. We introduced a very simple sort of wiring diagram in Section 2.2.2. These allowed us to draw a box

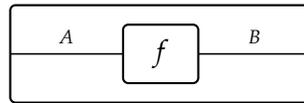


whenever $x_0 \leq x_1$. Chaining these together, we could prove facts in our poset. For example

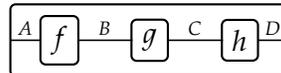


provides a proof that $x_0 \leq x_3$ (the exterior box) using three facts (the interior boxes), $x_0 \leq x_1$, $x_1 \leq x_2$, and $x_2 \leq x_3$.

As categorified posets, categories have basically the same sort of wiring diagram as posets—namely sequences of boxes inside a box. But since we have to replace the fact that $x_0 \leq x_1$ with the structure of a *set* of morphisms, we need to be able to label our boxes with morphism names:



Suppose given additional morphisms $g: A \rightarrow B$, and $h: C \rightarrow D$. Representing these each as boxes like we did for f , we might be tempted to stick them together to form a new box:



Ideally this would also be a morphism in our category: after all, we have said that we can represent morphisms with boxes with one input and one output. But wait, you say! We don't know which morphism it is. Is it $f.(g.h)$? Or $(f.g).h$? It's good that you are so careful. Luckily, we are saved by the properties that a category must have. Associativity says $f.(g.h) = (f.g).h$, so it doesn't matter which way we chose to try to decode the box.

Similarly, the identity morphism on an object x is drawn as on the left below, but we will see that it is not harmful to draw id_x in any of the following three ways:

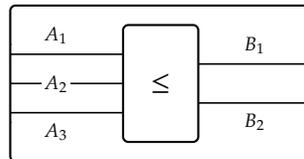


By Definition 3.2 the morphisms in a category satisfy two properties, called the unitality property and the associativity property. The unitality says that $\text{id}_x \cdot f = f = f \cdot \text{id}_y$ for any $f: x \rightarrow y$. In terms of diagrams this would say

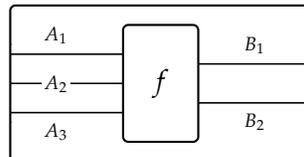
$$\boxed{\begin{array}{c} x \quad \boxed{} \quad x \quad \boxed{f} \quad y \\ \hline \end{array}} = \boxed{\begin{array}{c} x \quad \boxed{f} \quad y \\ \hline \end{array}} = \boxed{\begin{array}{c} x \quad \boxed{f} \quad y \quad \boxed{} \quad y \\ \hline \end{array}}$$

This means you can insert or discard any identity morphism you see in a wiring diagram. From this perspective, the coherence laws of a category—that is, the associativity law and the unitality law—are precisely what are needed to ensure we can draw these diagrams without ambiguity.

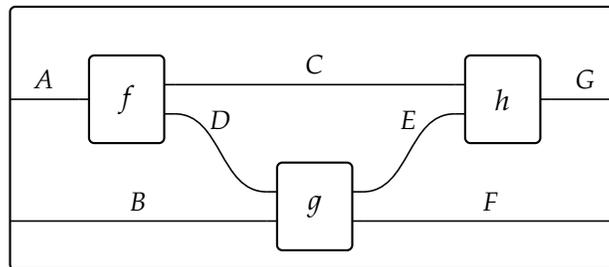
In Section 2.2.2, we also saw wiring diagrams for monoidal posets. Here we were allowed to draw boxes which can have multiple typed inputs and outputs, but with no label:



If we combine these ideas, we will obtain a categorification of symmetric monoidal posets: symmetric monoidal categories. A symmetric monoidal category is an algebraic structure in which we have labelled boxes with have multiple typed inputs and outputs:



Furthermore, a symmetric monoidal category has a composition rule and a monoidal product, which permit us to combine these boxes to interpret diagrams like this:



Finally, this structure must obey coherence laws, analogous to associativity and unitality in categories, that allow such diagrams to be unambiguously interpreted. In the next section we will be a bit more formal, but it is useful to keep in mind that, when we say our data must be well behaved, this is all we mean.

4.4.3 Monoidal categories

We defined \mathcal{V} -categories, for a symmetric monoidal poset \mathcal{V} in Definition 2.28. Just like posets turned out to be special kinds of categories (see Section 3.2.3), monoidal posets are special kinds of monoidal categories. And just like we can consider \mathcal{V} -categories for a monoidal poset, we can also consider \mathcal{V} -categories when \mathcal{V} is a monoidal category. This is another sort of categorification.

We will soon meet the monoidal category $(\mathbf{Set}, \{1\}, \times)$. The monoidal product will take two sets, S and T , and return the set $S \times T = \{(s, t) \mid s \in S, t \in T\}$. But whereas for monoidal posets we had the brute associative property $(p \otimes q) \otimes r = p \otimes (q \otimes r)$, the corresponding idea in \mathbf{Set} is not quite true:

$$\begin{aligned} S \times (T \times U) &:= \{(s, (t, u)) \mid s \in S, t \in T, u \in U\} \\ &\stackrel{?}{=} (S \times T) \times U := \{((s, t), u) \mid s \in S, t \in T, u \in U\}. \end{aligned}$$

They are slightly different sets: the first contains pairs consisting of an element in S and an element in $T \times U$, while the second contains pairs consisting of an element in $S \times T$ and an element in U . The sets are not equal, but they are clearly isomorphic, i.e. the difference between them is “just a matter of bookkeeping”. We thus need a structure—a bookkeeping isomorphism—to keep track of the associativity:

$$\alpha_{s,t,u}: \{(s, (t, u)) \mid s \in S, t \in T, u \in U\} \xrightarrow{\cong} \{((s, t), u) \mid s \in S, t \in T, u \in U\}.$$

There are a couple things to mention before we dive into these ideas. First, just because you replace brute things and properties with structures, it does not mean that you no longer have brute things and properties: new ones emerge! Not only that, but second, the new brute stuff tends to be more complex than what you started with. For example, above we replaced the associativity equation with an isomorphism $\alpha_{s,t,u}$, but now we need a more complex property to ensure that α behaves reasonably! The only way out of this morass is to add infinitely much structure, which leads one to “ ∞ -categories”, but we will not discuss that here.

Instead, we will continue with our categorification of monoidal posets, starting with a rough definition of symmetric monoidal categories. It’s rough in the sense that we suppress the technical bookkeeping, hiding it under the name “well behaved”.

Rough Definition 4.32. Let C be a category. A *symmetric monoidal structure* on C consists of the following constituents:

- (i) an object $I \in \text{Ob}(C)$ called the *monoidal unit*, and
- (ii) a functor $\otimes: C \times C \rightarrow C$, called the *monoidal product*

subject to well-behaved, natural isomorphisms

- (a) $\lambda_c: I \otimes c \cong c$ for every $c \in \text{Ob}(C)$,
- (b) $\rho_c: c \otimes I \cong c$ for every $c \in \text{Ob}(C)$,
- (c) $\alpha_{c,d,e}: (c \otimes d) \otimes e \cong c \otimes (d \otimes e)$ for every $c, d, e \in \text{Ob}(C)$, and

(d) $\sigma_{c,d}: c \otimes d \cong d \otimes c$ for every $c, d \in \text{Ob}(C)$, such that $\sigma \circ \sigma = \text{id}$.

A category equipped with a symmetric monoidal structure is called a *symmetric monoidal category*.

Remark 4.33. If the isomorphisms in (a), (b), and (c)—but *not* (d)—are replaced by equalities, then we say that the monoidal structure is *strict*, and this is a complete (non-rough) definition. In fact, symmetric strict monoidal categories are almost the same thing as symmetric monoidal categories. Ask a friendly expert category theorist to explain to you how!

Exercise 4.34. Check that monoidal categories generalize monoidal posets: a monoidal poset is a monoidal category $(\mathcal{P}, I, \otimes)$ where, for every $p, q \in \mathcal{P}$, the set $\mathcal{P}(p, q)$ has at most one element. \diamond

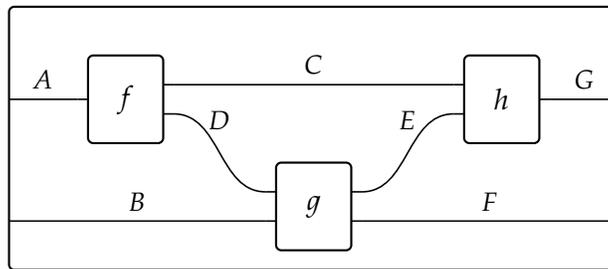
Example 4.35. As we said above, there is a monoidal structure on **Set** where the monoidal unit is some choice of singleton set, say $I := \{1\}$, and the monoidal product is $\otimes := \times$. What it means that \times is a functor is that:

- For any pair of objects, i.e. sets, $(S, T) \in \text{Ob}(\mathbf{Set} \times \mathbf{Set})$, one obtains a set $(S \times T) \in \text{Ob}(\mathbf{Set})$. We know what it is: the set of pairs $\{(s, t) \mid s \in S, t \in T\}$.
- For any pair of morphisms, i.e. functions, $f: S \rightarrow S'$ and $g: T \rightarrow T'$, one obtains a function $(f \times g): (S \times T) \rightarrow (S' \times T')$. It works pointwise: $(f \times g)(s, t) := (f(s), g(t))$.
- These should preserve identities: $\text{id}_S \times \text{id}_T = \text{id}_{S \times T}$ for any sets S, T .
- These should preserve composition: for any functions $S \xrightarrow{f} S' \xrightarrow{f'} S''$ and $T \xrightarrow{g} T' \xrightarrow{g'} T''$, one has

$$(f \times g).(f' \times g') = (f.g) \times (f'.g').$$

The four conditions, (a), (b), (c), and (d) give isomorphisms $\{1\} \times S \cong S$, etc. These maps are obvious in the case of **Set**, e.g. the function $\{(1, s) \mid s \in S\} \rightarrow S$ sending $(1, s)$ to s . We have been calling such things bookkeeping. \diamond

Exercise 4.36. Consider the monoidal category $(\mathbf{Set}, 1, \times)$, together with the diagram



Suppose that $A = B = C = D = F = G = \mathbb{Z}$ and $E = \mathbb{B} = \{\text{true}, \text{false}\}$, and suppose that $f_C(a) = |a|$, $f_D(a) = a * 5$, $g_E(d, b) = d \leq b$, $g_F(d, b) = d - b$, and $h(c, e) = \text{if } e \text{ then } c \text{ else } 1 - c$.

1. What are $g_E(5, 3)$ and $g_F(5, 3)$?
2. What are $g_E(3, 5)$ and $g_F(3, 5)$?

3. What is $h(5, \text{true})$?
4. What is $h(-5, \text{true})$?
5. What is $h(-5, \text{false})$?

The whole diagram now defines a function $A \times B \rightarrow G \times F$; call it q .

6. What are $q_G(-2, 3)$ and $q_F(-2, 3)$?
7. What are $q_G(2, 3)$ and $q_F(2, 3)$?

◇

We will see more monoidal categories throughout the remainder of this book.

4.4.4 Categories enriched in a symmetric monoidal category

We will not need this again, but had promised to explain why \mathcal{V} -categories, where \mathcal{V} is a symmetric monoidal poset, deserve to be seen as types of categories. The reason, as we have hinted, is that categories should really be called **Set**-categories. But wait, **Set** is not a poset! We'll have to generalize—categorify— \mathcal{V} -categories.

We now give a rough definition of categories enriched in a symmetric monoidal category \mathcal{V} . As in Definition 4.32, we suppress some technical parts in this sketch, hiding them under the name “usual associative and unital laws”.

Rough Definition 4.37. Let \mathcal{V} be a symmetric monoidal category, as in Definition 4.32. To specify a *category enriched in \mathcal{V}* , or a *\mathcal{V} -category*, denoted \mathcal{X} ,

- (i) one specifies a collection $\text{Ob}(\mathcal{X})$, elements of which are called *objects*;
- (ii) for every pair $x, y \in \text{Ob}(\mathcal{X})$, one specifies an object $\mathcal{X}(x, y) \in \mathcal{V}$, called the *hom-object* for x, y ;
- (iii) for every $x \in \text{Ob}(\mathcal{X})$, one specifies a morphism $\text{id}_x : I \rightarrow \mathcal{X}(x, x)$ in \mathcal{V} , called the *identity element*;
- (iv) for each $x, y, z \in \text{Ob}(\mathcal{X})$, one specifies a morphism $\cdot : \mathcal{X}(x, y) \otimes \mathcal{X}(y, z) \rightarrow \mathcal{X}(x, z)$, called the *composition morphism*.

These constituents are required to satisfy the usual associative and unital laws.

The precise, non-rough, definition can be found in other sources, e.g. [nLab], [con18], [Kel05].

Exercise 4.38. Recall from Example 4.35 that $\mathcal{V} = (\mathbf{Set}, \{1\}, \times)$ is a symmetric monoidal category. This means we can apply Definition 4.37. Does the (rough) definition roughly agree with the definition of category given in Definition 3.2? Or is there a subtle difference? ◇

Remark 4.39. We first defined \mathcal{V} -categories in Definition 2.28, where \mathcal{V} was required to be a monoidal poset. To check we're not abusing our terms, it's a good idea to make sure that \mathcal{V} -categories as per Definition 2.28 are still \mathcal{V} -categories as per Definition 4.37.

The first thing to observe is that every symmetric monoidal poset is a symmetric monoidal category (Exercise 4.34). So given a symmetric monoidal poset \mathcal{V} , we can apply Definition 4.37. The required data (i) and (ii) then get us off to a good start: both definitions of \mathcal{V} -category require objects and hom-objects, and they are specified in the

same way. On the other hand, Definition 4.37 requires two additional pieces of data: (iii) identity elements and (iv) composition morphisms. Where do these come from?

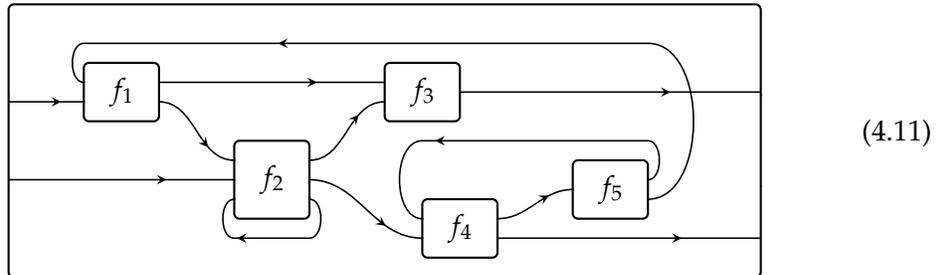
In the case of posets, there is at most one morphism between any two objects, so we do not need to choose an identity element and a composition morphism. Instead, we just need to make sure that an identity element and a composition morphism exist. This is exactly what properties (a) and (b) of Definition 2.28 say.

For example, the requirement (iii) that a \mathcal{V} -category \mathcal{X} has a chosen identity element $\text{id}_x: I \rightarrow \mathcal{X}(x, x)$ for the object x simply becomes the requirement (a) that $I \leq \mathcal{X}(x, x)$ is true in \mathcal{V} . This is typical of the story of categorification: what were mere properties in Definition 2.28 become structures in Definition 4.37.

Exercise 4.40. What are identity elements in Lawvere metric spaces (that is, **Cost**-categories)? How do we interpret this in terms of distances? \diamond

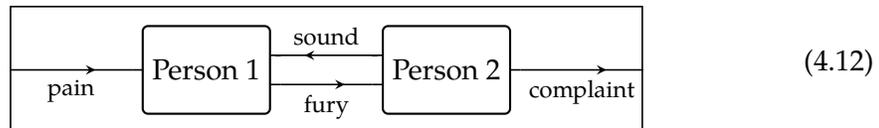
4.5 Profunctors form a compact closed category

In this section we will define compact closed categories and show that **Feas**, and more generally \mathcal{V} -profunctors, form such a thing. Compact-closed categories are monoidal categories whose wiring diagrams allow feedback. The wiring diagrams look like this:

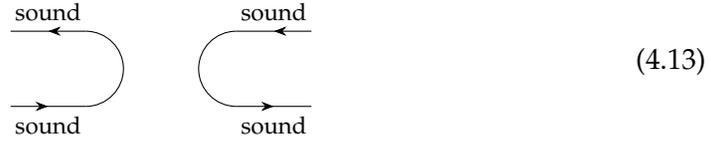


It’s been a while since we thought about co-design, but these were the kinds of wiring diagrams we drew, e.g. connecting the chassis, the motor, and the battery in Eq. (4.1). Compact closed categories are symmetric monoidal categories, with a bit more structure that allow us to formally interpret the sorts of feedback that occur in co-design problems. This same structure shows up in many other fields, including quantum mechanics and dynamical systems.

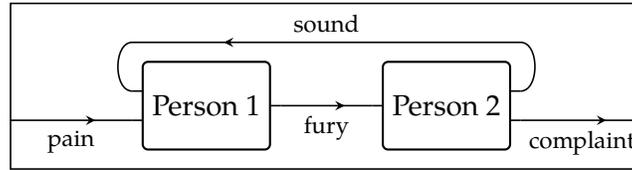
In Eq. (2.5) and Section 2.2.3 we discussed various flavors of wiring diagrams, including those with icons for splitting and terminating wires. For compact-closed categories, our additional icons allow us to bend outputs into inputs, and vice versa. To keep track of this, however, we draw arrows on our wire, which can either point forwards or backwards. For example, we can draw this



We then add icons—called a cap and a cup—allowing any wire to reverse direction from forwards to backwards and from backwards to forwards.



Thus we can draw the following



and its meaning is equivalent to that of Eq. (4.12).

We will begin by giving the axioms for a compact closed category. Then we will look again at feasibility relations in co-design—and more generally at enriched profunctors—and show that they indeed form a compact closed category.

4.5.1 Compact closed categories

As we said, compact closed categories are generalizations of symmetric monoidal categories (see Definition 4.32).

Definition 4.41. Let (C, I, \otimes) be a symmetric monoidal category, and $c \in \text{Ob}(C)$ an object. A *dual* for c consists of three constituents

- (i) an object $c^* \in \text{Ob}(C)$, called the *dual of c* ,
- (ii) a morphism $\eta_c: I \rightarrow c^* \otimes c$, called the *unit for c* ,
- (iii) a morphism $\epsilon_c: c \otimes c^* \rightarrow I$, called the *counit for c* .

These are required to satisfy two equations for every $c \in \text{Ob}(C)$, which we draw as commutative diagrams:

$$\begin{array}{ccc}
 c & \xlongequal{\quad} & c \\
 \cong \downarrow & & \uparrow \cong \\
 c \otimes I & & I \otimes c \\
 c \otimes \eta_c \downarrow & & \uparrow \epsilon_c \otimes c \\
 c \otimes (c^* \otimes c) & \xrightarrow{\cong} & (c \otimes c^*) \otimes c
 \end{array}
 \qquad
 \begin{array}{ccc}
 c^* & \xlongequal{\quad} & c^* \\
 \cong \downarrow & & \uparrow \cong \\
 I \otimes c^* & & c^* \otimes I \\
 \eta_c \otimes c^* \downarrow & & \uparrow c^* \otimes \epsilon_c \\
 (c^* \otimes c) \otimes c^* & \xrightarrow{\cong} & c^* \otimes (c \otimes c^*)
 \end{array}
 \tag{4.14}$$

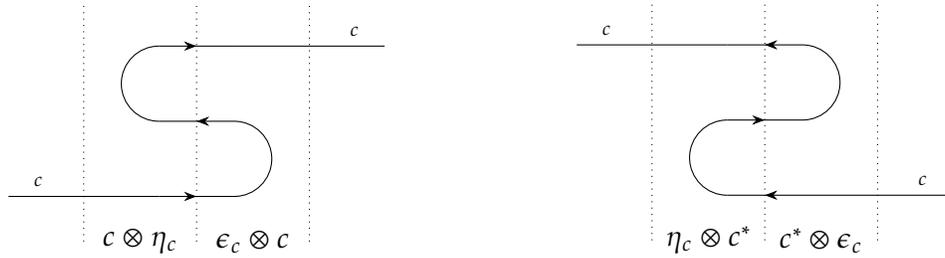
These equations are sometimes called the *snake equations*.

If for every object $c \in \text{Ob}(C)$ there exists a dual c^* for c , then we say that (C, I, \otimes) is *compact closed*.

In a compact closed category, each wire is equipped with a direction. For any object c , a forward-pointing wire labeled c is considered equivalent to a backward-pointing wire labeled c^* , i.e. \xrightarrow{c} is the same as $\xleftarrow{c^*}$. The cup and cap discussed above are in fact the unit and counit morphisms; they are drawn as follows.



In wiring diagrams, the snake equations (4.14) are then drawn as follows:



Note that the pictures in Eq. (4.13) correspond to ϵ_{sound} and η_{sound^*} .

Recall the notion of monoidal closed poset; a monoidal category can also be monoidal closed. This means that for every pair of objects $c, d \in \text{Ob}(C)$ there is an object $c \multimap d$ and an isomorphism $C(b \otimes c, d) \cong C(b, c \multimap d)$, natural in b . While we will not prove it here, compact closed categories are so-named because they are a special type of monoidal closed category.

Proposition 4.42. *If C is a compact closed category, then*

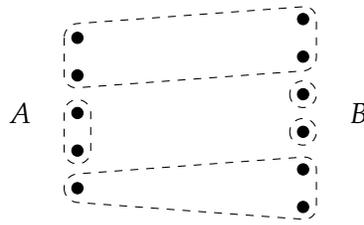
1. C is monoidal closed;
- and for any object $c \in \text{Ob}(C)$,
2. if c^* and c' are both duals to c then there is an isomorphism $c^* \cong c'$; and
 3. there is an isomorphism between c and its double-dual, $c \cong c^{**}$.

Before returning to co-design, we give another example of a compact closed category, called **Corel**, which we'll see again in the next two chapters.

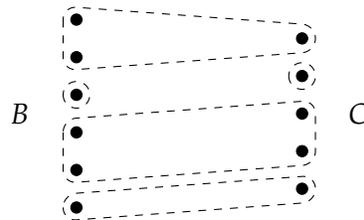
Example 4.43. Recall, from Definition 1.10, that an equivalence relation on a set A is a reflexive, symmetric, and transitive binary relation on A . Given two finite sets, A and B , a *corelation* $A \rightarrow B$ is an equivalence relation on $A \sqcup B$.

So, for example, here is a corelation from a set A having five elements to a set B having six elements; two elements are equivalent if they are encircled by the same

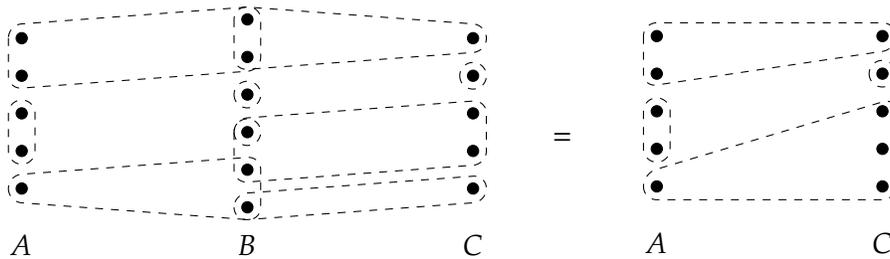
dashed line.



There exists a category, denoted **Corel**, where the objects are finite sets, and where a morphism from $A \rightarrow B$ is a corelation $A \rightarrow B$. The composition rule is simpler to look at than to write down formally.² If in addition to the corelation $\alpha: A \rightarrow B$ above we have another corelation $\beta: B \rightarrow C$



Then the composite $\beta \circ \alpha$ of our two corelations is given by



That is, two elements are equivalent in the composite corelation if we may travel from one to the other staying within equivalence classes of either α or β .

The category **Corel** may be equipped with the symmetric monoidal structure (\emptyset, \sqcup) . This monoidal category is compact closed, with every finite set its own dual. Indeed, note that for any finite set A there is an equivalence relation on $A \sqcup A := \{(a, 1), (a, 2) \mid a \in A\}$ where each part simply consists of the two elements $(a, 1)$ and $(a, 2)$ for each $a \in A$. The unit on a finite set A is the corelation $\eta_A: \emptyset \rightarrow A \sqcup A$ specified by this equivalence relation; similarly the counit on A is the corelation $\epsilon_A: A \sqcup A \rightarrow \emptyset$ specified by this same equivalence relation. \blacklozenge

² To compose corelations $\alpha: A \rightarrow B$ and $\beta: B \rightarrow C$, we need to construct an equivalence relation $\alpha.\beta$ on $A \sqcup C$. To do so requires three steps: (i) consider α and β as relations on $A \sqcup B \sqcup C$, (ii) take the transitive closure of their union, and then (iii) restrict to an equivalence relation on $A \sqcup C$. Here is the formal description. Note that as binary relations, we have $\alpha \subseteq (A \sqcup B) \times (A \sqcup B)$, and $\beta \subseteq (B \sqcup C) \times (B \sqcup C)$. We also have three inclusions: $\iota_{A \sqcup B}: A \sqcup B \rightarrow A \sqcup B \sqcup C$, $\iota_{B \sqcup C}: B \sqcup C \rightarrow A \sqcup B \sqcup C$, and $\iota_{A \sqcup C}: A \sqcup C \rightarrow A \sqcup B \sqcup C$. Recalling our notation from Section 1.5, we define

$$\alpha.\beta := \iota_{A \sqcup C}^*((\iota_{A \sqcup B})!(\alpha) \vee (\iota_{B \sqcup C})!(\beta)).$$

Exercise 4.44. Consider the set $\underline{3} = \{1, 2, 3\}$.

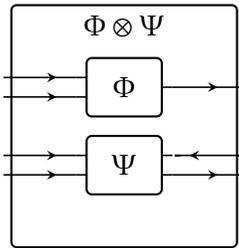
1. Draw a picture of the unit corelation $\emptyset \rightarrow \underline{3} \sqcup \underline{3}$.
2. Draw a picture of the counit corelation $\underline{3} \sqcup \underline{3} \rightarrow \emptyset$.
3. Check that the snake equations (4.14) hold. (Since every object is its own dual, you only need to check one of them.) ◇

4.5.2 Feas as a compact closed category

We close the chapter by returning to co-design and showing that **Feas** has a compact closed structure. This is what allows us to draw the kinds of wiring diagrams we saw in Eqs. (4.1), (4.11), and (4.12): it is what puts actual mathematics behind these pictures.

Instead of just detailing this compact closed structure for **Feas**, it’s no extra work to prove that for any skeletal (commutative) quantale $(\mathcal{V}, I, \otimes)$ the profunctor category $\mathbf{Prof}_{\mathcal{V}}$ is compact closed, so we’ll prove this general fact. Indeed, all we need to do is construct a monoidal structure and duals for objects.

Monoidal products in $\mathbf{Prof}_{\mathcal{V}}$ are just product categories In terms of wiring diagrams, the monoidal structure looks like stacking wires or boxes on top of one another, with no new interaction.



We take our monoidal product on $\mathbf{Prof}_{\mathcal{V}}$ to be that given by the product of \mathcal{V} -categories; the definition was given in Definition 2.51, and we worked out several examples there. To recall, the formula for the hom-sets in $\mathcal{X} \times \mathcal{Y}$ is given by

$$(\mathcal{X} \times \mathcal{Y})((x, y), (x', y')) := \mathcal{X}(x, x') \otimes \mathcal{Y}(y, y').$$

But monoidal products need to be given on morphisms also, and the morphisms in $\mathbf{Prof}_{\mathcal{V}}$ are \mathcal{V} -profunctors. So given \mathcal{V} -profunctors $\Phi: \mathcal{X}_1 \rightarrow \mathcal{X}_2$ and $\Psi: \mathcal{Y}_1 \rightarrow \mathcal{Y}_2$, one defines a \mathcal{V} -profunctor $(\Phi \times \Psi): \mathcal{X}_1 \times \mathcal{Y}_1 \rightarrow \mathcal{X}_2 \times \mathcal{Y}_2$ by

$$(\Phi \times \Psi)((x_1, y_1), (x_2, y_2)) := \Phi(x_1, x_2) \otimes \Psi(y_1, y_2).$$

The monoidal unit in $\mathbf{Prof}_{\mathcal{V}}$ is $\mathbf{1}$ To define a monoidal structure on $\mathbf{Prof}_{\mathcal{V}}$, we need not only a monoidal product—as defined above—but also a monoidal unit. Recall the \mathcal{V} -category $\mathbf{1}$; it has one object, say 1 , and $\infty(1, 1) = I$ is the monoidal unit of \mathcal{V} . We take $\mathbf{1}$ to be the monoidal unit of $\mathbf{Prof}_{\mathcal{V}}$.

Exercise 4.45. In order for $\mathbf{1}$ to be a monoidal unit, there are supposed to be isomorphisms $\mathcal{X} \times \mathbf{1} \rightarrow \mathcal{X}$ and $\mathbf{1} \times \mathcal{X} \rightarrow \mathcal{X}$, for any \mathcal{V} -category \mathcal{X} . What are they? ◇

Duals in $\mathbf{Prof}_{\mathcal{V}}$ are just opposite categories In order to regard $\mathbf{Prof}_{\mathcal{V}}$ as a compact closed category (Definition 4.41), it remains to specify duals and the corresponding cup and cap.

Duals are easy: for every \mathcal{V} -category \mathcal{X} , its dual is its opposite category \mathcal{X}^{op} (see Exercise 2.50). The unit and counit then look like identities. To elaborate, the unit is a \mathcal{V} -profunctor $\eta_{\mathcal{X}}: \mathbf{1} \rightarrow \mathcal{X}^{\text{op}} \times \mathcal{X}$. By definition, this is a \mathcal{V} -functor

$$\eta_{\mathcal{X}}: \mathbf{1} \times \mathcal{X}^{\text{op}} \times \mathcal{X} \rightarrow \mathcal{V};$$

we define it by $\eta_{\mathcal{X}}(1, x, x') := \mathcal{X}(x, x')$. Similarly, the counit is the profunctor $\epsilon_{\mathcal{X}}: (\mathcal{X} \times \mathcal{X}^{\text{op}}) \rightarrow \mathbf{1}$, defined by $\epsilon_{\mathcal{X}}(x, x', 1) := \mathcal{X}(x, x')$.

Exercise 4.46. Check these proposed units and counits do indeed obey the snake equations Eq. (4.14). ◇

4.6 Summary and further reading

This chapter introduced three important ideas in category theory: profunctors, categorification, and monoidal categories. Let's talk about them in turn.

Profunctors generalize binary relations. In particular, we saw that the idea of profunctor over a monoidal poset gave us the additional power necessary to formalize the idea of a feasibility relation between resource posets. The idea of a feasibility relation is due to Andrea Censi; he called them *monotone codesign problems*. The basic idea is explained in [Cen15], where he also gives a programming language to specify and solve codesign problems. In [Cen17], Censi further discusses how to use estimation to make solving codesign problems computationally efficient.

We also saw profunctors over the poset \mathbf{Cost} , and how to think of these as bridges between Lawvere metric space. We referred earlier to Lawvere's paper [Law73]; plenty more on \mathbf{Cost} -profunctors can be found there.

Profunctors, however are vastly more general than the two examples of have discussed; \mathcal{V} -profunctors can be defined not only when \mathcal{V} is a poset, but for any symmetric monoidal category. A delightful, detailed exposition of profunctors and related concepts such as equipments, companions and conjoiners, symmetric monoidal bicategories can be found in [Shu08; Shu10].

We have not defined symmetric monoidal bicategories, but you would be correct if you guessed this is a sort of categorification of symmetric monoidal categories. Baez and Dolan tell the subtle story of categorifying categories to get ever *higher* categories in [BD98]. Crane and Yetter give a number of examples of categorification in [CY96].

Finally, we talked about monoidal categories and compact closed categories. Monoidal categories are a classic, central topic in category theory, and a quick introduction can be found in [Mac98]. Wiring diagrams play a huge role in this book and in applied category theory in general; while in informal use for years, these were first formalized in the case of monoidal categories. You can find the details here [JS93; JSV96].

Compact closed categories are a special type of structured monoidal category; there are many others. For a broad introduction to the different flavors of monoidal category, detailed through their various styles of wiring diagram, see [Sel10].

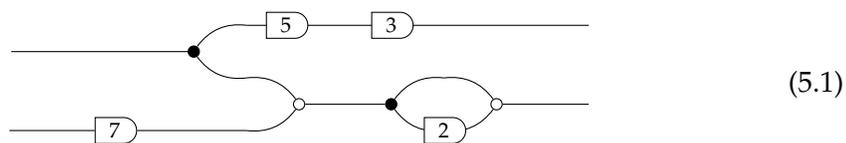
Signal flow graphs: Props, presentations, and proofs

5.1 Comparing systems as interacting signal processors

Cyber-physical systems are systems that involve tightly interacting physical and computational parts. An example is an autonomous car: sensors inform a decision system that controls a steering unit that drives a car. While such systems involve complex interactions of many different types—both physical ones, such as the driving of a wheel by a motor, or a voltage placed across a wire, and computational ones, such as a program that takes a measured velocity and returns a desired acceleration—it is often useful to abstract and model the system as simply passing around and processing signals. For this illustrative sketch, we will just think of signals as things which we can add and multiply, such as real numbers.

Interaction in cyber-physical systems can often be understood as variable sharing; i.e. when two systems are linked, certain variables become shared. For example, when we connect two train carriages by a physical coupling, the train carriages must have the same velocity, and their positions differ by a constant. Similarly, when we connect two electrical ports, the electric potentials at these two ports now must be the same, and the current flowing into one must equal the current flowing out of the other.

Note that both these examples involve the physical joining of two systems; more figuratively, we might express the interconnection by drawing a line connecting boxes that represent the systems. In its simplest form, this is captured by the formalism of signal flow graphs, due to Shannon in the 1940s. Here is an example of a signal flow graph.



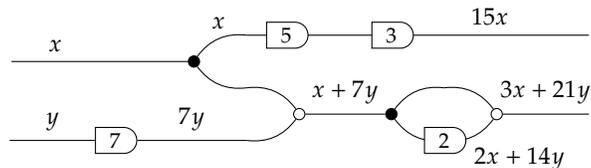
We consider the dangling wires on the left as inputs, and those on the right as outputs.

In Eq. (5.1) we see three types of signal processing units:

- Each unit labelled by a number a takes an input and multiplies it by a .
- Each black dot takes an input and produces two copies of it.
- Each white dot takes two inputs and produces their sum.

Thus the above signal flow graph takes in two input signals, say x (on the upper left wire) and y (on the lower left wire), and—going from left to right as described above—produces two output signals: $u = 15x$ (upper right) and $v = 3x + 21y$ (lower right).

Let's show some of the steps (leaving others off to avoid clutter):



In words, the signal flow graph first multiplies y by 7, then splits x into two copies, adds the second copy of x to the lower signal to get $x + 7y$, and so on.

A signal flow graph might describe an existing system, or it might specify a system to be built. In either case, it is important to be able to analyze these diagrams to understand how the composite system converts inputs to outputs. This is reminiscent of a co-design problem from Chapter 4, which asks how to evaluate the composite feasibility relation from a diagram of feasibility relations. We can use this process of evaluation to determine whether two different signal flow graphs in fact specify the same composite system, and hence whether a system meets a given specification.

In this chapter, however, we introduce categorical tools—props and presentations—for reasoning more directly with the diagrams. Recall from Chapter 2 that symmetric monoidal posets are a type of symmetric monoidal category where the *morphisms* are constrained to be very simple: there can be at most one morphism between any two objects. Here shall see that signal flow graphs form an example of morphisms in a different, complementary simplification of symmetric monoidal category known as a prop. A prop is a symmetric monoidal category where the *objects* are constrained to be very simple: the objects are the natural numbers, and the monoidal product on objects is given by addition. Just as the wiring diagrams for symmetric monoidal posets did not require labels on the boxes, wiring diagrams for props do not require labels on the wires. This makes props particularly suited for describing diagrammatic formalisms such as signal flow graphs, which only have wires of a single type.

Finally, many systems behave in what is called a *linear* way, and the foundation of control theory—a branch of engineering that underlies the study of cyber-physical systems—is the study of linear systems. Similarly, linear algebra is a foundational part of modern mathematics, both pure and applied, which underlies not just control theory, but also the practice of computing, physics, statistics, and many others. Matrices are of course central to linear algebra. As we analyze signal flow graphs, we shall see that

they are in fact a way of recasting matrix algebra in graphical terms; more formally, we shall say that signal flow graphs have *functorial semantics* as matrices.

5.2 Props and presentations

Signal flow graphs as in Eq. (5.1) are easily seen to be wiring diagrams of some sort. However they have the property that, unlike for monoidal posets and monoidal categories, there is no need to label the wires. This corresponds to a form of symmetric monoidal category, known as a prop, which has a very particular set of objects.

5.2.1 Props: definition and first examples

Recall the definition of strict symmetric monoidal category from Definition 4.32 and Remark 4.33.

Definition 5.1. A *prop* is a strict symmetric monoidal category $(C, 0, +)$ for which $\text{Ob}(C) = \mathbb{N}$, the monoidal unit is $0 \in \mathbb{N}$, and the monoidal product on objects is given by addition.

Since the objects of a prop are always the natural numbers, to specify a prop P we specify five things:

- (i) a set $P(m, n)$ of morphisms $m \rightarrow n$, for $m, n \in \mathbb{N}$.
- (ii) for all $n \in \mathbb{N}$, an identity map $\text{id}_n: n \rightarrow n$.
- (iii) for all $m, n \in \mathbb{N}$, a symmetry map $\sigma_{m,n}: m + n \rightarrow n + m$.
- (iv) a composition rule: given $f: m \rightarrow n$ and $g: n \rightarrow p$, a map $f.g: m \rightarrow p$.
- (v) a monoidal product on morphisms: given $f: m \rightarrow m'$ and $g: n \rightarrow n'$, we provide $(f + g): m + n \rightarrow m' + n'$.

Once one specifies the above data, he should check that their specifications satisfy the rules of symmetric monoidal categories (see Definition 4.32).

Example 5.2. There is a prop **FinSet** where the morphisms $f: m \rightarrow n$ are functions from $\underline{m} = \{1, \dots, m\}$ to $\underline{n} = \{1, \dots, n\}$. (The identities, symmetries, and composition rule are obvious.) The monoidal product on functions is given by the disjoint union of functions: that is, given $f: \underline{m} \rightarrow \underline{m'}$ and $g: \underline{n} \rightarrow \underline{n'}$, we define $f + g: \underline{m + n} \rightarrow \underline{m' + n'}$ by

$$i \mapsto \begin{cases} f(i) & \text{if } 1 \leq i \leq m; \\ m' + g(i) & \text{if } m + 1 \leq i \leq m + n. \end{cases} \quad (5.2) \quad \blacklozenge$$

Exercise 5.3. In Example 5.2 we said that the identities, symmetries, and composition rule in **FinSet** “are obvious”. In math lingo, this just means “we trust the reader can figure them out if she spends the time tracking down the definitions and fitting them together.”

1. Draw two morphism $f: 3 \rightarrow 2$ and $g: 2 \rightarrow 4$ in **FinSet**.
2. Draw $f + g$.

3. What is the composition rule for morphisms $f: m \rightarrow n$ and $g: n \rightarrow p$ in **FinSet**?
4. What are the identities in **FinSet**? Draw some.
5. Pick some $m, n \in \mathbb{N}$, and draw the symmetric map $\sigma_{m,n}$ in **FinSet**? ◇

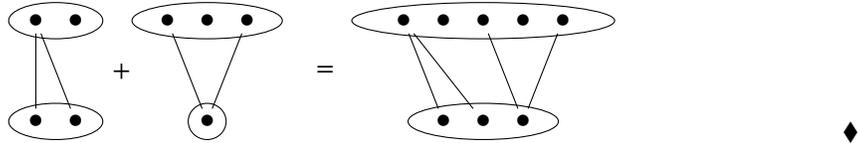
Example 5.4. There is a prop **Bij** where the morphisms $f: m \rightarrow n$ are bijections $\underline{m} \rightarrow \underline{n}$. Note that in this case morphisms $m \rightarrow n$ only exist when $m = n$; when $m \neq n$ the homset **Bij**(m, n) is empty. We call morphisms in **Bij** *permutations*. Since **Bij** is a subcategory of **FinSet**, we can define the monoidal product to be as in Eq. (5.2). ◇

Example 5.5. The compact closed category **Corel**, in which the morphisms $f: m \rightarrow n$ are partitions on $\underline{m} \sqcup \underline{n}$ (see last chapter), is a prop. ◇

Example 5.6. There is a prop **Rel_{Fin}** for which morphisms $m \rightarrow n$ are relations, $R \subseteq \underline{m} \times \underline{n}$. The composition of R with $S \subseteq \underline{n} \times \underline{p}$ is

$$R.S := \{(i, k) \in \underline{m} \times \underline{p} \mid \exists(j \in \underline{n}). (i, j) \in R \text{ and } (j, k) \in S\}.$$

The monoidal product is relatively easy to formalize using universal properties,¹ but one might get better intuition from pictures:



Exercise 5.7. A posetal prop is a prop that is also a poset. That is, a posetal prop is a symmetric monoidal poset of the form (\mathbb{N}, \leq) , for some preorder \leq on \mathbb{N} , where the monoidal product on objects is addition. We’ve spent a lot of time discussing order structures on the natural numbers. Give three examples of a posetal prop. ◇

Definition 5.8. Let \mathcal{C} and \mathcal{D} be props. A functor $F: \mathcal{C} \rightarrow \mathcal{D}$ is called a *prop functor* if

- (a) F is identity-on-objects, i.e. $F(n) = n$ for all $n \in \text{Ob}(\mathcal{C}) = \text{Ob}(\mathcal{D}) = \mathbb{N}$, and
- (b) for all $f: m_1 \rightarrow m_2$ and $g: n_1 \rightarrow n_2$ in \mathcal{C} , we have $F(f) + F(g) = F(f + g)$ in \mathcal{D} .

Example 5.9. The inclusion $i: \mathbf{Bij} \rightarrow \mathbf{FinSet}$ is a prop functor. Perhaps more interestingly, there is a prop functor $F: \mathbf{FinSet} \rightarrow \mathbf{Rel}_{\mathbf{Fin}}$. It sends a function $f: \underline{m} \rightarrow \underline{n}$ to the relation $F(f) := \{(i, j) \mid f(i) = j\} \subseteq \underline{m} \times \underline{n}$. ◇

5.2.2 The prop of port graphs

An important example of a prop is the one in which morphisms are open, directed, acyclic port graphs, as we next define. We will just call them port graphs.

Definition 5.10. For $m, n \in \mathbb{N}$, an (m, n) -port graph (V, in, out, ι) is specified by

- a set V , elements of which are called *vertices*,

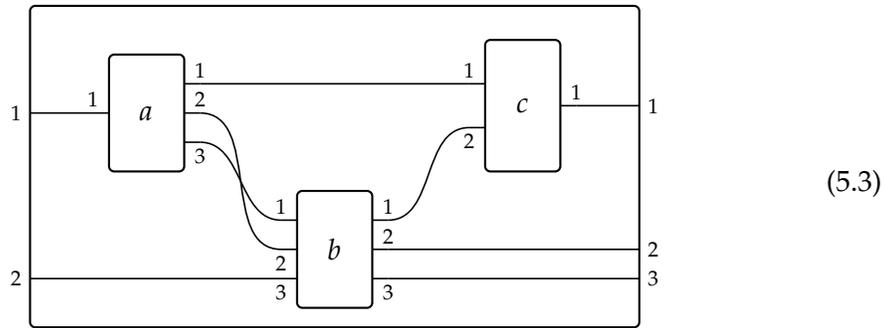
¹The monoidal product $R_1 + R_2$ of relations $R_1 \subseteq m_1 \times n_1$ and $R_2 \subseteq m_2 \times n_2$ is given by $R_1 \sqcup R_2 \subseteq (m_1 \times n_1) \sqcup (m_2 \times n_2) \subseteq (m_1 \sqcup m_2) \times (n_1 \sqcup n_2)$.

- two functions $in, out: V \rightarrow \mathbb{N}$, and
- a bijection $\iota: \underline{m} \sqcup O \xrightarrow{\cong} I \sqcup \underline{n}$, where $I = \{(v, i) \mid v \in V, 1 \leq i \leq in(v)\}$ is the set of *vertex inputs*, and $O = \{(v, i) \mid v \in V, 1 \leq i \leq out(v)\}$ is the set of *vertex outputs*.

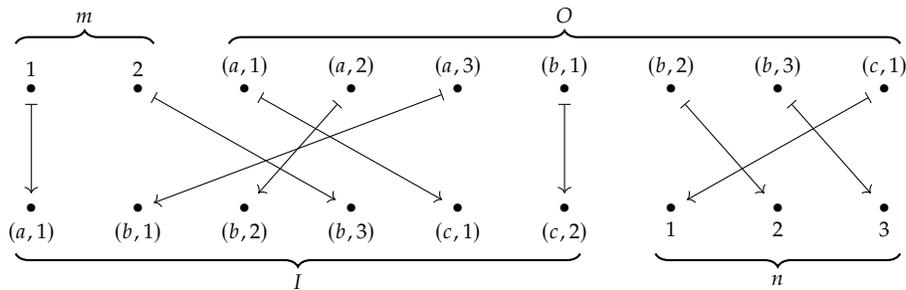
This data must obey the following acyclicity condition. First, use the bijection ι to construct the *internal flow graph* with vertices V and with an arrow $e_{v,j}^{u,i}: u \rightarrow v$ for every $i, j \in \mathbb{N}$ such that $\iota(u, i) = (v, j)$. If the internal flow graph is acyclic—that is, if the only path from any vertex v to itself is the empty path—then we say that (V, in, out, ι) is a port graph.

This seems quite a technical construction, but it’s quite intuitive once you unpack it a bit. Let’s do this.

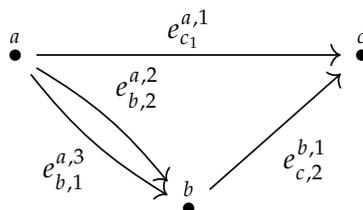
Example 5.11. Here is an example of a $(2, 3)$ -port graph, i.e. with $m = 2$ and $n = 3$:



Since the port graph has type $(2, 3)$, we draw two ports on the left hand side of the outer box, and three on the right. The vertex set is $V = \{a, b, c\}$ and, for example $in(a) = 1$ and $out(a) = 3$, so we draw one port on the left-hand side and three ports on the right-hand side of the box labelled a . The bijection ι is what tells us how the ports are connected by wires:



The internal flow graph—which one can see is acyclic—is shown below:



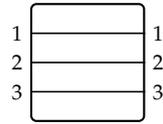
As you might guess from the picture 5.3, port graphs are closely related to wiring diagrams for monoidal categories, and even more closely related to wiring diagrams for props. \blacklozenge

A category \mathbf{PG} whose morphisms are port graphs Given an (m, n) -port graph (V, in, out, ι) and an (n, p) -port graph (V', in', out', ι') , we may compose them to define an (m, p) -port graph $(V \sqcup V', [in, in'], [out, out'], \iota'')$. Here $[in, in']$ denotes the function $V \sqcup V' \rightarrow \mathbb{N}$ which maps elements of V according to in , and elements of V' according to in' , and similarly for $[out, out']$. The bijection $\iota'' : \underline{m} \sqcup O \sqcup O' \rightarrow I \sqcup I' \sqcup \underline{p}$ is defined as follows:

$$\iota''(x) = \begin{cases} \iota(x) & \text{if } \iota(x) \in I \\ \iota'(\iota(x)) & \text{if } \iota(x) \in \underline{n} \\ \iota'(x) & \text{if } x \in O'. \end{cases}$$

Exercise 5.12. Describe how port graph composition looks, with respect to the visual representation of Example 5.11, and give a nontrivial example. \blacklozenge

We thus have a category \mathbf{PG} , whose objects are natural numbers $\text{Ob}(\mathbf{PG}) = \mathbb{N}$, whose morphisms are port graphs $\mathbf{PG}(m, n) = \{(V, in, out, \iota) \mid \text{as in Definition 5.10}\}$. Composition of port graphs is as above, and the identity port graph on n is the (n, n) -port graph $(\emptyset, !, !, id_n)$, where $! : \emptyset \rightarrow \mathbb{N}$ is the unique such function. The identity on an object, say 3, is depicted as follows:



The monoidal structure structure on \mathbf{PG} This category \mathbf{PG} is in fact a prop. The monoidal product of two port graphs $G := (V, in, out, \iota)$ and $G' := (V', in', out', \iota')$ is given by taking the disjoint union of ι and ι' :

$$G + G' := ((V \sqcup V'), [in, in'], [out, out'], (\iota \sqcup \iota')). \tag{5.4}$$

The monoidal unit is $(\emptyset, !, !, !)$.

5.2.3 Free constructions and universal properties

Given some sort of categorical structure, like a poset, a category, or a prop, it is useful to be able to construct one according to your own specification. (This should not be surprising.) The minimally-constrained structure that contains all the data you specify is called the *free structure* on your specification. We have already seen some examples of free structures.

Example 5.13 (The free poset on a relation). For posets, we saw the construction of taking the reflexive, transitive closure on a relation. That is, given a relation $R \subseteq P \times P$,

the reflexive, transitive closure of R is called the free poset on R . Rather than specify all the inequalities in the poset (P, \leq) , we can specify just a few inequalities $p \leq q$, and let our “closure machine” add in the minimum number of other inequalities necessary to make P a poset. To obtain a poset out of a graph, or Hasse diagram, we consider a graph (V, A, s, t) as defining a relation $\{(s(a), t(a)) \mid a \in A\} \subseteq V \times V$, and apply this machine.

But in what sense is the reflexive, transitive closure of a relation $R \subseteq P \times P$ really the *minimally-constrained* poset containing R ? One way of understanding this is that the extra equalities impose no further constraints when defining a monotone map *out* of P . We are claiming that freeness has something to do with maps *out*! As strange as an asymmetry might seem here (one might ask, “why not maps in?”), the reader will have an opportunity to explore for herself in Exercises 5.14 and 5.15. \blacklozenge

Exercise 5.14. Let P be a set, let $R \subseteq P \times P$ a relation, let (P, \leq_P) be the poset obtained by taking the reflexive, transitive closure of R , and let (Q, \leq_Q) be an arbitrary poset. Finally, let $f: P \rightarrow Q$ be a function, not assumed monotone.

1. Suppose that for every $x, y \in P$, if $R(x, y)$ then $f(x) \leq f(y)$. Show that f defines a monotone map $f: (P, \leq_P) \rightarrow (Q, \leq_Q)$.
2. Suppose that f defines a monotone map $f: (P, \leq_P) \rightarrow (Q, \leq_Q)$. Show that for every $x, y \in P$, if $R(x, y)$ then $f(x) \leq f(y)$.

We call this the *universal property* of the free poset. \blacklozenge

Exercise 5.15. Let P, R , etc. be as in Exercise 5.14. We want to see that the universal property is really about maps out of—and not maps in to—the reflexive, transitive closure (P, \leq) . So let $g: Q \rightarrow P$ be a function.

1. Suppose that for every $a, b \in Q$, if $a \leq b$ then $(g(a), g(b)) \in R$. Is it true that g defines a monotone map $g: (Q, \leq_Q) \rightarrow (P, \leq_P)$?
2. Suppose that g defines a monotone map $g: (Q, \leq_Q) \rightarrow (P, \leq_P)$. Is it true that for every $a, b \in Q$, if $a \leq b$ then $(g(a), g(b)) \in R$?

The lesson is that maps between structured objects are defined to preserve structure. This means the domain of a map must be more constrained than the codomain. Thus having the fewest additional constraints coincides with having the most maps out—every function that respects our generating constraints should define a map.² \blacklozenge

Example 5.16 (The free category on a graph). There is a similar story for categories. Indeed, we saw in Section 3.2.1 the construction of the path category from a graph. There we specified a set V of objects, and then a set A of arrows, or morphisms we wanted in our category. Each arrow had a source and target, a.k.a. domain and codomain, given by functions $s, t: A \rightarrow V$. Since this data can be interpreted as a graph, we were easily able to visualize it. The free category, which we also called the path category, on this data has the set V as its set of objects, and as morphisms $u \rightarrow v$ paths in this graph. As such, it is a category that contains the elements of A as

²A higher-level justification understands freeness as a left adjoint—of an adjunction often called the syntax/semantics adjunction—but we will not discuss that here.

morphisms, but obeys no equations other than the usual ones that categories obey, like having identity maps that obey the unit law. \blacklozenge

Exercise 5.17. Let $G = (V, A, s, t)$ be a graph, and let \mathcal{G} be the path category on G . Given another category C , show that the set of functors $\mathcal{G} \rightarrow C$ are in one-to-one correspondence with the set of pairs of functions (f, q) , where $f: V \rightarrow \text{Ob}(C)$ and q is a function from A to morphisms of C such that $q(a) \in C(f(s(a)), f(t(a)))$ for all a . \blacklozenge

Exercise 5.18 (The free monoid on a set). Recall that monoids are one-object categories. For any set A , there is a graph with one vertex and an arrow from the vertex to itself for each $a \in A$. Thus we can construct a free monoid from just the data of a set A .

1. What is the free monoid on the set $A = \{a\}$?
2. What is the free monoid on the set $A = \{a, b\}$? \blacklozenge

5.2.4 Free props

We have been discussing free constructions, in particular for posets and categories. A similar construction exists for props. Since we already know what the objects of the prop will be—the natural numbers—all we need to specify is a set G of *generating morphisms*, together with the arities,³ that we want to be in our prop. From this information we can generate the free prop on G . Just like free posets and free categories, the free prop is characterized by a universal property in terms of maps out.

To be more precise, suppose we have a set G together with functions $s, t: G \rightarrow \mathbb{N}$. The free prop on G —assuming it exists—should be the prop \mathcal{G} such that, for any prop C , the prop functors $\mathcal{G} \rightarrow C$ are in one-to-one correspondence with functions $G \rightarrow C$ that send each $g \in G$ to a morphism $s(g) \rightarrow t(g)$ in C . This description completely specifies \mathcal{G} (up to isomorphism), but we still need to show such a prop exists. We now give an explicit construction in terms of port graphs (see Definition 5.10).

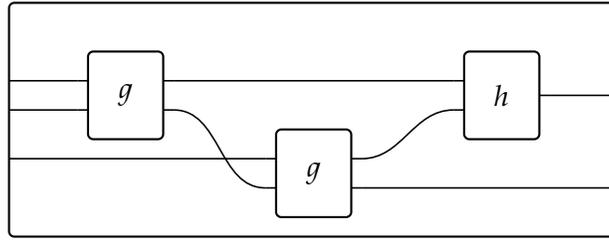
Definition 5.19. Suppose we have a set G and functions $s, t: G \rightarrow \mathbb{N}$. The free prop $\mathbf{Free}(G)$ is defined as follows. For any $m, n \in \mathbb{N}$, a morphism $m \rightarrow n$ in $\mathbf{Free}(G)$ is a G -labeled (m, n) -port graph, i.e. a pair (Γ, ℓ) , where $\Gamma = (V, in, out, \iota)$ is an (m, n) -port graph and $\ell: V \rightarrow G$ is a function, such that the arities agree: $s(\ell(v)) = in(v)$ and $t(\ell(v)) = out(v)$. We call ℓ the *labeling function*.

Composition and the monoidal structure are just those for port graphs \mathbf{PG} , see Eq. (5.4).

The morphisms in $\mathbf{Free}(G)$ are G -labeled port graphs (V, in, out, ι) as in Definition 5.10. To draw a port graph, just as in Example 5.11, we draw each vertex $v \in V$ as a box with $in(v)$ -many ports on the left and $out(v)$ -many vertices on the right. In Definition 5.19, we added a labeling function $\ell: V \rightarrow G$, and so we add these labels to the picture. Note that multiple boxes can be labelled with the same generator. For

³The arity of a prop morphism is a pair $(m, n) \in \mathbb{N} \times \mathbb{N}$, where m is the number of inputs and n is the number of outputs.

example, if $G = \{f: 1 \rightarrow 1, g: 2 \rightarrow 2, h: 2 \rightarrow 1\}$, then the following is a morphism in $\mathbf{Free}(G)$:



Note that in this morphism the generator g is used twice, while the generator f is not used at all. This is perfectly fine.

Example 5.20. The free prop on the empty set \emptyset is **Bij**. This is because each morphism must have a labelling function of the form $V \rightarrow \emptyset$, and hence we must have $V = \emptyset$. Thus the only morphisms (n, m) are those given by port graphs $(\emptyset, !, !, \sigma)$, where $\sigma: n \rightarrow m$ is a bijection. \blacklozenge

Exercise 5.21. Show that the free prop on the set $\{\rho_{m,n}: m \rightarrow n \mid m, n \in \mathbb{N}\}$, i.e. having one generating morphism for each $(m, n) \in \mathbb{N}^2$, is the prop of port graphs. \blacklozenge

An alternate way to obtain $\mathbf{Free}(G)$ Port graphs provide a convenient formalism of thinking about morphisms in the free prop on a set G , but there is another approach which is also useful. It is syntactic, in the sense that we start with a small stock of basic morphisms, including G , and then we inductively build new morphisms from them using the basic operations of props: namely composition and monoidal product. Sometimes the conditions of monoidal categories—e.g. associativity, unitality, functoriality, see Definition 4.32—force two such morphisms to be equal, and so we dutifully equate them. When we are done, the result is again the free prop $\mathbf{Free}(G)$. Let's make this more formal.

Definition 5.22. Suppose we have a set G and functions $s, t: G \rightarrow \mathbb{N}$. We define a G -generated prop expression $e: m \rightarrow n$, where $m, n \in \mathbb{N}$, inductively as follows:

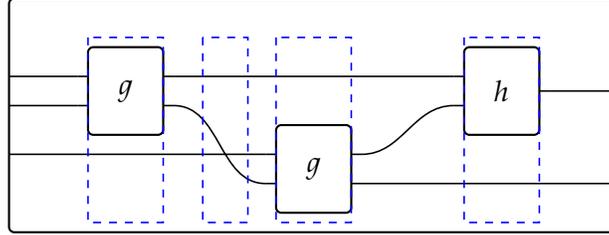
- The empty morphism $\text{id}_0: 0 \rightarrow 0$, the identity morphism $\text{id}_1: 1 \rightarrow 1$, and the symmetry $\sigma: 2 \rightarrow 2$ are expressions.⁴
- the generators $g \in G$ are expressions $g: s(g) \rightarrow t(g)$.
- if $\alpha: m \rightarrow n$ and $\beta: p \rightarrow q$ are expressions, then $\alpha + \beta: m + p \rightarrow n + q$ is an expression.
- if $\alpha: m \rightarrow n$ and $\beta: n \rightarrow p$ are expressions, then $\alpha.\beta: m \rightarrow p$ is an expression.

We write $\text{Expr}(G)$ for the set of expressions in G . If $e: m \rightarrow n$ is a prop expression, we refer to (m, n) as its *arity*.

So both G -labeled port graphs and G -generated prop expressions describe morphisms in the free prop $\mathbf{Free}(G)$. Here's how to turn a port graph into a prop expression:

⁴One can think of σ as the "swap" icon $\times: 2 \rightarrow 2$

imagine a vertical line moving through the port graph from left to right. Whenever you see “action”—either a box or wires crossing—write down the sum (using +) of all the boxes g , all the symmetries σ , and all the wires id_1 in that column. Finally, compose all of those action columns. For example, in the picture below we see four action columns:



Here the result is $(g + \text{id}_1).(\text{id}_1 + \sigma).(\text{id}_1 + g).(h + \text{id}_1)$.

Exercise 5.23. Consider again the free prop on generators $G = \{f: 1 \rightarrow 1, g: 2 \rightarrow 2, h: 2 \rightarrow 1\}$. Draw a picture of $(f + \text{id}_1 + \text{id}_1).(\sigma + \text{id}_1).(\text{id}_1 + h).\sigma.g$. \diamond

5.2.5 Props via presentations

In Section 3.2.2 we saw that a database schema consists of a graph together with imposed equations between paths. Similarly here, sometimes we want to construct a prop whose morphisms obey specific equations. But rather than mere paths, the things we want to equate are prop expressions as in Definition 5.22.

Rough Definition 5.24. A *presentation* (G, s, t, E) for a prop is a set G , functions $s, t: G \rightarrow \mathbb{N}$, and a set $E \subseteq \text{Expr}(G) \times \text{Expr}(G)$ of pairs of G -generated prop expressions, such that e_1 and e_2 have the same arity for each $(e_1, e_2) \in E$. We refer to G as the set of generators and to E as the set of *equations* in the presentation.⁵

The prop \mathcal{G} *presented* by this presentation is the prop where morphism are elements in $\text{Expr}(G)$, quotiented by both the equations $e_1 = e_2$ where $(e_1, e_2) \in E$, and by the axioms of strict symmetric monoidal categories.

Remark 5.25. Given a presentation (G, s, t, E) , it can be shown that the prop \mathcal{G} has a universal property in terms of “maps out”. Namely prop functors from \mathcal{G} to any other prop \mathcal{C} are in one-to-one correspondence with functions f from G to the set of morphisms in \mathcal{C} such that

- for all $g \in G$, $f(g)$ is a morphism $s(g) \rightarrow t(g)$, and
- for all $(e_1, e_2) \in E$, we have that $f(e_1) = f(e_2)$ in \mathcal{C} , where $f(e)$ denotes the morphism in \mathcal{C} obtained by applying f to each generators in the expression e , and then composing the result in \mathcal{C} .

Exercise 5.26. Is it the case that the free prop on generators (G, s, t) , defined in Definition 5.19, is the same thing as the prop presented by (G, s, t, \emptyset) , having no relations? Or is there a subtle difference somehow? \diamond

⁵Elements of E , which we call equations, are traditionally called “relations”.

5.3 Simplified signal flow graphs

We now return to signal flow graphs, expressing them in terms of props. We will discuss a simplified form without feedback (the only sort we have discussed so far), and then extend to the usual form of signal flow graphs in Section 5.4.3. But before we can do that, we must say what we mean by signals; this gets us into the algebraic structure of “rigs”. We will get to signal flow graphs in Section 5.3.2.

5.3.1 Rigs

Signals can be amplified, and they can be added. Adding and amplification interact via a distributive law, as follows: if we add two signals, and then amplify them by some amount a , it should be the same as amplifying the two signals separately by a , then adding the results.

We can think of all the possible amplifications as forming a structure called a rig,⁶ defined as follows.

Definition 5.27. A *rig* is a tuple $(R, 0, +, 1, *)$, where R is a set, $0, 1 \in R$ are elements, and $+, *: R \times R \rightarrow R$ are functions, such that

- (a) $(R, +, 0)$ is a commutative monoid,
- (b) $(R, *, 1)$ is a monoid, and
- (c) $a * (b + c) = a * b + a * c$ and $(a + b) * c = a * c + b * c$ for all $a, b, c \in R$.
- (d) $a * 0 = 0 = 0 * a$ for all $a \in R$.

We have already encountered many examples of rigs.

Example 5.28. The natural numbers form a rig $(\mathbb{N}, 0, +, 1, *)$. ♦

Example 5.29. The Booleans form a rig $(\mathbb{B}, \text{false}, \vee, \text{true}, \wedge)$. ♦

Example 5.30. Any quantale $\mathcal{V} = (V, \leq, I, \otimes)$ determines a rig $(V, 0, \vee, I, \otimes)$, where $0 = \bigvee \emptyset$ is the empty join. See Definition 2.55. ♦

Example 5.31. If R is a rig and $n \in \mathbb{N}$ is any natural number, then the set $\text{Mat}_n(R)$ of $(n \times n)$ -matrices in R forms a rig. Such a matrix $M \in \text{Mat}_n(R)$ can be thought of as a function $M: (\underline{n} \times \underline{n}) \rightarrow R$. Then addition $M + N$ is given by $(M + N)(i, j) := M(i, j) + N(i, j)$ and multiplication $M * N$ is given by $(M * N)(i, j) := \sum_{k \in \underline{n}} M(i, k) * N(k, j)$. The 0-matrix is $0(i, j) := 0$. ♦

Exercise 5.32. We said in Example 5.31 that for any rig R , the set $\text{Mat}_n(R)$ forms a rig. What is its multiplicative identity $1 \in \text{Mat}_n(R)$? ♦

The following is an example for readers who are familiar with the algebraic structure known as “rings”.

Example 5.33. Any ring forms a rig. In particular, the real numbers $(\mathbb{R}, 0, +, 1, *)$ are a rig. The difference between a ring and rig is that a ring, in addition to all the properties

⁶Rigs are also known as *semi-rings*.

of a rig, must also have additive inverses, or *negatives*. A common mnemonic is that a rig is a ring without negatives. ♦

5.3.2 The iconography of signal flow graphs

A signal flow graph is supposed to keep track of the amplification, by elements of a rig R , to which signals are subjected. While not strictly necessary,⁷ we will assume the signals themselves are elements of the same rig. We call elements of the rig *signals* for the time being.

Amplification of a single signal by some value $a \in R$ is simply depicted like so:



We interpret the above icon as depicting a system where a signal enters at the wire on the left, is multiplied by a , and is output at the wire on the right.

What is more interesting than just a single signal amplification, however, is the interaction of signals. There are four other important icons in signal flow graphs.



Let's go through them one by one. The first two are old friends from Chapter 2: copy and discard.



We interpret this diagram as taking in an input signal on the left, and outputting that same value to both wires on the right. It is basically the “copy” operation from Section 2.2.3.

Next, we have the ability to discard signals.



This takes in any signal, and outputs nothing. It is basically the “waste” operation from Section 2.2.3.

Next, we have the ability to add signals.



This takes the two input signals and adds them, to produce a single output signal.

Finally, we have the zero signal.

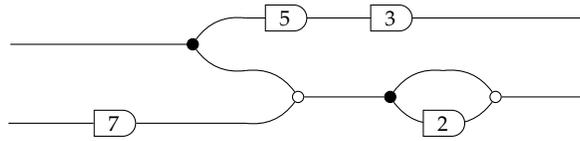


This has no inputs, but always outputs the 0 element of the rig.

⁷The necessary requirement for the material below to make sense is that the signals take values in an R -module M . We will not discuss this here, keeping to the simpler requirement that $M = R$.

Using these icons, we can build more complex signal flow graphs. To compute the operation performed by a signal flow graph we simply trace the paths with the above interpretations, plugging outputs of one icon into the inputs of the next icon.

For example, consider the rig $R = \mathbb{N}$ from Example 5.28, where the scalars are the natural numbers. Recall the signal flow graph from Eq. (5.1) in the introduction:

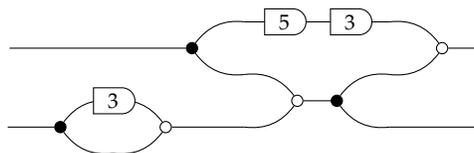


As we explained, this takes in two input signals x and y , and returns two output signals $a = 15x$ and $b = 3x + 21y$.

In addition to tracing the processing of the values as they move forward through the graph, we can also calculate these values by summing over paths. More explicitly, to get the contribution of a given input wire to a given output wire, we take the sum, over all paths p , of the total amplification along that path. So, for example, there is one path from the top input to the top output. On this path, the signal is first copied, which does not affect its value, then amplified by 5, and finally amplified by 3. Thus, if x is the first input signal, then this contributes $15x$ to the first output. Since there is no path from the bottom input to the top output (one is not allowed to traverse paths backwards), the signal at the first output is exactly $15x$.

Both inputs contribute to the bottom output. In fact, each input contributes in two ways, as there are two paths to it from each input. The top input thus contributes $3x = x + 2x$, whereas the bottom input, passing through an additional $*7$ amplification, contributes $21y$.

Exercise 5.34. The flow graph



takes in two natural numbers, x and y , and produces two output signals. What are they? \diamond

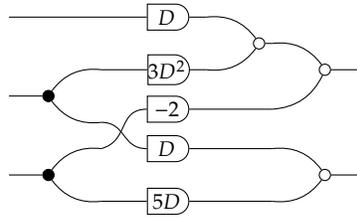
Example 5.35. This example is for those who have some familiarity with differential equations. A linear system of differential equations provides a simple way to specify the movement of a particle. For example, consider a particle whose position (x, y, z) in 3-dimensional space is determined by the following equations:

$$\begin{aligned} \dot{x} + 3\dot{y} - 2z &= 0 \\ \dot{y} + 5\dot{z} &= 0 \end{aligned}$$

Using what is known as the Laplace transform, one can convert this into a linear system involving a formal variable D , which stands for “differentiate”. Then the system becomes

$$\begin{aligned} Dx + 3D^2y - 2z &= 0 \\ Dy + 5Dz &= 0 \end{aligned}$$

which can be represented by the signal flow graph



Signal flow graphs as morphisms in a free prop We can formally define simplified signal flow graphs using props.

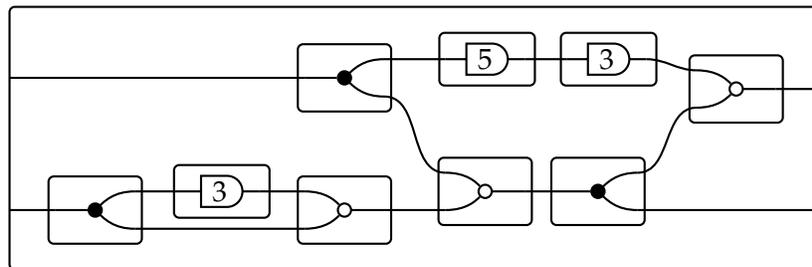
Definition 5.36. Let R be a rig (see Definition 5.27). Consider the set

$$G_R := \left\{ \begin{array}{c} \text{---} \circ \text{---} \\ \text{---} \bullet \text{---} \end{array} \right\} \cup \left\{ \text{---} \boxed{a} \text{---} \mid a \in R \right\},$$

and let $s, t: G_R \rightarrow \mathbb{N}$ be given by the number of dangling wires on the left and right of the generator icon respectively. A *simplified signal flow graph* is a morphism in the free prop $\mathbf{Free}(G_R)$ on this set G_R of generators. We define $\mathbf{SFG}_R := \mathbf{Free}(G_R)$.

For now we’ll drop the term ‘simplified’, since these are the only sort of signal flow graph we know. We’ll return to signal flow graphs in their full glory—i.e. including feedback—in Section 5.4.3.

Example 5.37. To be more in line with our representations of both wiring diagrams and port graphs, as a morphism in $\mathbf{Free}(G_R)$, the signal flow graph from Exercise 5.34 could be drawn as follows:



But it is easier on the eye to draw this as in Exercise 5.34, and so we’ll draw our diagrams in that fashion.

More importantly, props also provide language to understand the semantics of signal flow graphs. Although the signal flow graphs themselves are free props, their semantics—their meaning in our model of signals flowing—will arise when we add equations to our props, as in Definition 5.24. These equations will tell us when two signal flow graphs act the same way on signals. For example,

(5.5)

both express the same behavior: a single input signal is copied twice so that three identical copies of the input signal are output.

If two signal flow graphs S, T are almost the same, with the one exception being that somewhere we replace the left-hand side of Eq. (5.5) with the right-hand side, then S and T have the same behavior. But there are other replacements we could make to a signal flow graph that do not change its behavior. Our next goal is to find a complete description of these replacements.

5.3.3 The prop of matrices over a rig

Signal flow graphs are closely related to matrices. In previous chapters we used matrices with values in a quantale \mathcal{V} —a closed monoidal poset with all joins—to represent systems of interrelated points and connections between them, such as profunctors. The quantale gave us the structure and axioms we needed in order for matrix multiplication to work properly. But we know from Example 5.30 that quantales are examples of rigs, and in fact matrix multiplication makes sense in any rig R . In Example 5.31, we explained how that works for $(n \times n)$ -matrices in R , showing that $\text{Mat}_n(R)$ forms a new rig, for any choice of $n \in \mathbb{N}$. But we can do better, assembling all the $(m \times n)$ -matrices into a single prop.

An $m \times n$ -matrix M with values in R is a function $M: \underline{m} \times \underline{n} \rightarrow R$. Given an $m \times n$ -matrix M and an $n \times p$ -matrix N , their *composite* is the $m \times p$ -matrix $M.N$ defined by

$$M.N(a, c) = \sum_{b \in \underline{n}} M(a, b) \times N(b, c),$$

for any $a \in \underline{m}$ and $c \in \underline{p}$. Here the $\sum_{b \in \underline{n}}$ just means repeated addition (using the rig R 's $+$ operation), as usual.

Remark 5.38. Conventionally, one generally considers a matrix A acting on a vector v by multiplication in the order Av , where v is a column vector. In keeping with our composition convention, we use the opposite order, $v.A$, where v as a row vector. See for example Eq. (5.6) for when this is implicitly used.

Definition 5.39. Let R be a rig. We define the *prop of R -matrices*, denoted $\mathbf{Mat}(R)$, to be the prop whose morphisms $m \rightarrow n$ are the $m \times n$ -matrices with values in R .

Composition of morphisms is given by matrix multiplication as above. The monoidal product is given by the direct sum of matrices: given matrices $A: m \rightarrow n$ and $b: p \rightarrow q$, we define $A + B: m + p \rightarrow n + q$ to be the block matrix

$$\begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix}$$

where each 0 represents a matrix of zeros of the appropriate dimension ($m \times q$ and $n \times p$). We refer to any combination of multiplication and direct sum as a *interconnection* of matrices.

Exercise 5.40. Let A and B be the following matrices with values in \mathbb{N} :

$$A = \begin{pmatrix} 3 & 3 & 1 \\ 2 & 0 & 4 \end{pmatrix} \qquad B = \begin{pmatrix} 2 & 5 & 6 & 1 \end{pmatrix}$$

What is the monoidal product matrix $A + B$? ◇

5.3.4 Turning signal flow graphs into matrices

Let's now consider more carefully what we mean by the meaning, or *semantics*, of each signal flow graph. We'll use matrices.

In the examples above, the signal output on a given wire is given by a sum of amplified input values. If we can only measure the input and output signals, and care nothing for what happens in between, then each signal flow graph may as well be reduced to a matrix of amplifications. For example, let's call the inputs in Eq. (5.1) x and y , and the outputs a and b , as we did above. We can represent the signal flow graph of Eq. (5.1) by either the matrix on the left (for more detail) or the matrix on the right if the labels are clear from context:

$$\begin{array}{c|cc} & a & b \\ \hline x & 15 & 3 \\ y & 0 & 21 \end{array} \qquad \begin{pmatrix} 15 & 3 \\ 0 & 21 \end{pmatrix}$$

Every signal flow graph can be interpreted as a matrix The generators G_R from Definition 5.36 are shown again in the table below. Each can easily be interpreted as a matrix. For example, we can interpret amplification by $a \in R$ as the 1×1 matrix $(a): 1 \rightarrow 1$: it is an operation that takes an input $x \in R$ and returns $a * x$. Similarly, we can interpret \triangleright as the 2×1 matrix $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$: it is an operation that takes two inputs, x and y , and returns $x + y$. Here is a table showing the interpretation of each generator.

generator	icon	matrix	arity
amplify by $a \in R$		(a)	$1 \rightarrow 1$
add		$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$	$2 \rightarrow 1$
zero		$()$	$0 \rightarrow 1$
copy		$(1 \ 1)$	$1 \rightarrow 2$
discard		$()$	$1 \rightarrow 0$

(5.6)

Note that both zero and discard are represented by empty matrices, but of differing dimensions. In linear algebra it is unusual to consider matrices of the form $0 \times n$ or $n \times 0$ for various n to be different, but it really the same story: you can multiply a 0×3 matrix by a $3 \times n$ matrix for any n , but you can not multiply it by a $2 \times n$ matrix.

Since signal flow graphs are morphisms in a free prop, the table in 5.6 is enough to show that we can interpret any signal flow diagram as a matrix.

Theorem 5.41. *There is a prop functor $S: \mathbf{SFG}_R \rightarrow \mathbf{Mat}(R)$ that sends the generators $g \in G$ icons to the matrices as described in Table 5.6.*

Proof. This follows immediately from the universal property of free props, Remark 5.25. □

We have now constructed a matrix $S(g)$ from any signal flow graph g . But what is this matrix? Both for the example signal flow graph in Eq. (5.1) and for the generators in Definition 5.36, the associated matrix is $m \times n$, where m is the number of inputs and n the number of outputs, with (i, j) th entry describing the amplification of the i th input that contributes to the j th output. This is how one would hope or expect the functor S to work in general; but does it? We have used a big hammer—the universal property of free constructions—to obtain our functor S . Our next goal is to check that it works in the expected way. Doing so is a matter of using induction over the set of prop expressions, as we now see.

Proposition 5.42. *Let g be a signal flow graph with m inputs and n outputs. The matrix $S(g)$ is the $m \times n$ -matrix with (i, j) -entry describing the amplification of the i th input that contributes to the j th output.*

Proof. Recall from Definition 5.22 that a G_R -generated prop expression is built from the morphisms $\text{id}_0: 0 \rightarrow 0$, $\text{id}_1: 1 \rightarrow 1$, $\sigma: 2 \rightarrow 2$, and the generators in G_R , using the following two rules:

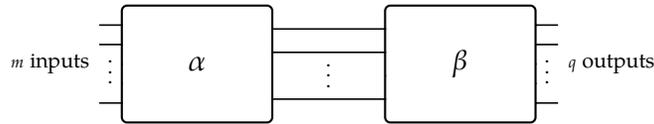
- if $\alpha: m \rightarrow n$ and $\beta: p \rightarrow q$ are expressions, then $\alpha + \beta: m + p \rightarrow n + q$ is an expression.
- if $\alpha: m \rightarrow n$ and $\beta: n \rightarrow p$ are expressions, then $\alpha.\beta: m \rightarrow p$ is an expression.

S is a prop functor by Theorem 5.41, which by Definition 5.8 must preserve identities, compositions, monoidal products, and symmetries. Thus the proposition is true when g is equal to id_0 , id_1 , and σ . Indeed, the empty signal flow graph $\text{id}_0: 0 \rightarrow 0$ must be sent to the unique (empty) matrix $(): 0 \rightarrow 0$. The morphisms id_1 , σ , and $a \in R$ map to the identity matrix, the swap matrix, and the scalar matrix (a) respectively:

$$\text{---} \mapsto (1) \quad \text{and} \quad \text{X} \mapsto \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \text{and} \quad \text{---} \boxed{a} \text{---} \mapsto (a)$$

In each case, the (i, j) -entry gives the amplification of the i th input to the j th output. Thus it remains to show that if the proposition holds for $\alpha: m \rightarrow n$ and $\beta: p \rightarrow q$, then it holds for (i) $\alpha.\beta$ (when $n = p$) and (ii) $\alpha + \beta$.

To prove (i), consider the following picture of $\alpha.\beta$:

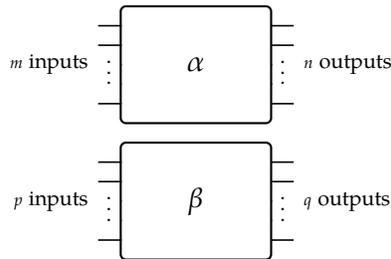


Here $\alpha: m \rightarrow n$ and $\beta: n \rightarrow q$ are signal flow graphs, assumed to obey the proposition. Consider the i th input and k th output of $\alpha.\beta$; we'll just call these i and k . The amplification that the i contributes to k is the sum—over all paths from i to k —of the amplification along that path. So let's also fix some $j \in \underline{n}$, and consider paths from i to k that run through j . By distributivity, the total amplification from i to k is the total amplification over all paths from i to j times the total amplification over all paths from j to k . Since all paths from i to k must run through some j th output of α /input of β , the amplification that i contributes to j is

$$\sum_{j \in \underline{n}} \alpha(i, j) * \beta(j, k).$$

This is exactly the formula for matrix multiplication, so $\alpha.\beta$ obeys the proposition when α and β do.

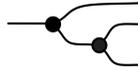
Proving (ii) is more straightforward. The monoidal product $\alpha + \beta$ of signal flow graphs looks like this:



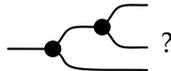
No new paths are created; the only change is to reindex the inputs and outputs. In particular, the i th input of α is the i th input of $\alpha + \beta$, the j th output of α is the j th output of $\alpha + \beta$, the i th input of β is the $(m + i)$ th output of $\alpha + \beta$, and the j th output of β is the

$(n + j)$ th output of $\alpha + \beta$. This means that the matrix with (i, j) th entry describing the amplification of the i th input that contributes to the j th output is $S(\alpha) + S(\beta) = S(\alpha + \beta)$, as in Definition 5.39. This proves the proposition. \square

Exercise 5.43. What matrices does the signal flow graph



represent? What about the signal flow graph



Are they equal? \diamond

5.3.5 The idea of functorial semantics

Let's pause for a moment to reflect on what we have just learned. First, signal flow diagrams are the morphisms in a prop. This means we have two special operations we can do to form new signal flow diagrams from old, namely composition and monoidal product, i.e. serial and parallel. We might think of this as specifying a 'grammar' for the 'language' of signal flow diagrams.

We have also seen that each signal flow diagram can be interpreted, or given semantics, as a matrix. Moreover, matrices have the same grammatical structure: they form a prop, and we can construct new matrices from old using composition and the monoidal product. Finally, in Theorem 5.41 we completed this picture by showing that semantic interpretation is a prop functor between the prop of signal flow graphs and the prop of matrices. Thus we say that matrices give *functorial semantics* for signal flow diagrams.

Functorial semantics is an key manifestation of compositionality. It says that the matrix meaning $S(g)$ for a big signal flow graph g can be computed by:

1. splitting g up into little pieces,
2. computing the very simple matrices for each piece, and
3. using matrix multiplication and direct sum to put the pieces back together to obtain the desired meaning, $S(g)$.

This functoriality is useful in practice, for example in speeding up computation of the semantics of signal flow graphs: for large signal flow graphs, composing matrices is much faster than tracing paths.

5.4 Graphical linear algebra

In this section we will begin to develop something called graphical linear algebra, which extends the ideas above. This formalism is actually quite powerful. For example,

with it we can easily and *graphically* prove certain conjectures from control theory that, although they were eventually solved, required fairly elaborate matrix algebra arguments [FSR16].

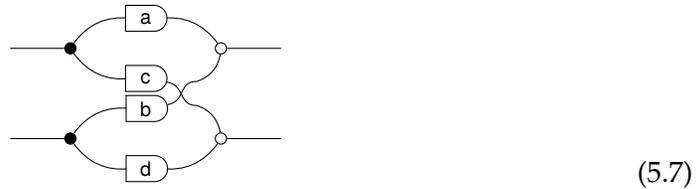
5.4.1 A presentation of $\mathbf{Mat}(R)$

Let R be a rig, as defined in Definition 5.27. The main theorem of the previous section, Theorem 5.41, provided a functor $S: \mathbf{SFG}_R \rightarrow \mathbf{Mat}(R)$ that converts any signal flow graph into a matrix. Next we show that S is “full”: that any matrix can be represented by a signal flow graph.

Proposition 5.44. *Given any matrix $M \in \mathbf{Mat}(R)$, there exists a signal flow graph $g \in \mathbf{SFG}_R$ such that $S(g) = M$.*

sketch. Let $M \in \mathbf{Mat}(R)$ be an $(m \times n)$ -matrix. We want a signal flow graph g such that $S(g) = M$. In particular, to compute $S(g)(i, j)$, we know that we can simply compute the amplification the i th input contributes to the j th output. The key idea then is to construct g so that there is exactly one path from i th input to the j th output, and that this path has exactly one scalar multiplication icon, namely $M(i, j)$.

The general construction is a little technical (see Exercise 5.46), but the idea is clear from just considering the case of 2×2 -matrices. Suppose M is the 2×2 -matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$. Then we define g to be the signal flow graph



Tracing paths, it is easy to see that $S(g) = M$. Note that g is the composite of four layers, each layer respectively a monoidal product of (i) copy maps, (ii) scalar multiplications, (iii) swaps and identities, (iv) addition maps.

For the general case, see Exercise 5.46. □

Exercise 5.45. Draw signal flow graphs that represent the following matrices:

$$1. \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix} \qquad 2. \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \qquad 3. \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \qquad \diamond$$

Exercise 5.46. Write down a detailed proof of Proposition 5.44. Suppose M is an $m \times n$ -matrix. Follow the idea of the (2×2) -case in Eq. (5.7), and construct the signal flow graph g —having m inputs and n outputs—as the composite of four layers, respectively comprising (i) copy maps, (ii) scalars, (iii) swaps and identities, (iv) addition maps. ◇

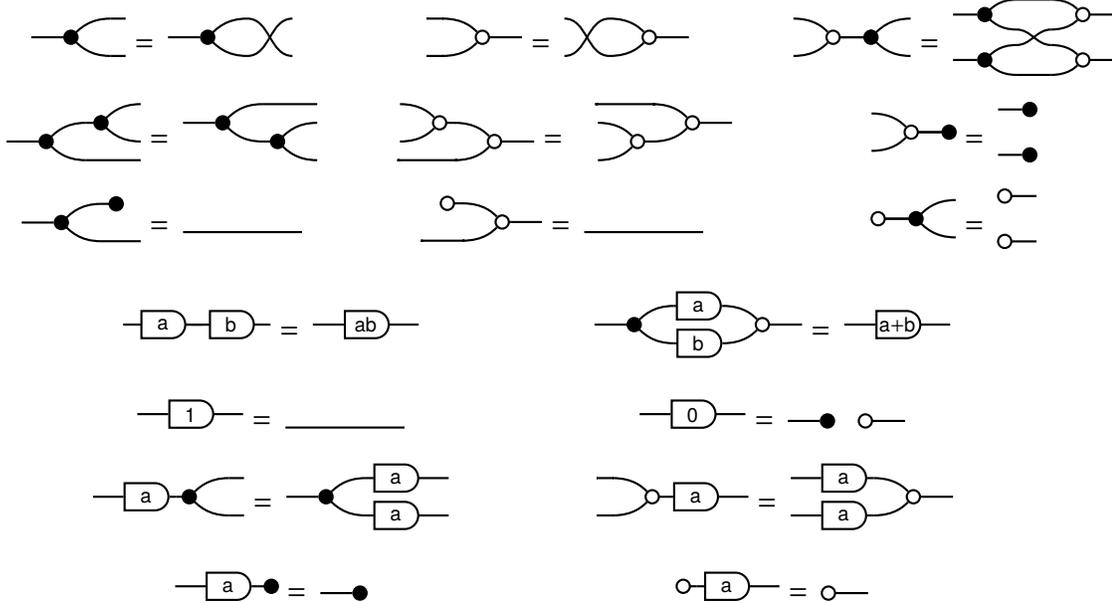
We can also use Proposition 5.44 and its proof to give a presentation of $\mathbf{Mat}(R)$.

Theorem 5.47. *The prop $\mathbf{Mat}(R)$ is isomorphic to the prop with the following presentation. The set of generators is the set*

$$G_R := \left\{ \text{cup}, \text{cap}, \text{cross}, \text{dot} \right\} \cup \left\{ \boxed{a} \mid a \in R \right\},$$

the same as the set of generators for \mathbf{SFG}_R ; see Definition 5.36.

The equations are

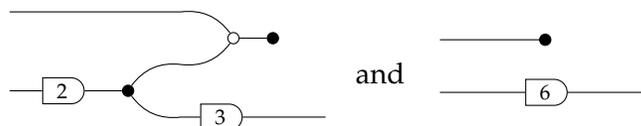


where $a, b \in R$.

Proof. The key idea is that these equations are sufficient to rewrite any G_R -generated prop expression into the form used in the proof of Proposition 5.44, and hence enough to show the equality of any two expressions that represent the same matrix. Details can be found in [BE15] or [BS17]. \square

Sound and complete presentation of matrices Once you get used to it, Theorem 5.47 provides an intuitive, visual way to reason about matrices. Indeed, the theorem implies two signal flow graphs represent the same matrix if and only if one can be turned into the other by local application of the above equations and the prop axioms. In the jargon of logic, we call this a sound and complete reasoning system. To be more specific, *sound* refers to the forward direction of the above statement: two signal flow graphs represent the same matrix if one can be turned into the other using the given rules. *Complete* refers to the reverse direction: if two signal flow graphs represent the same matrix, then we can prove it using these rules.

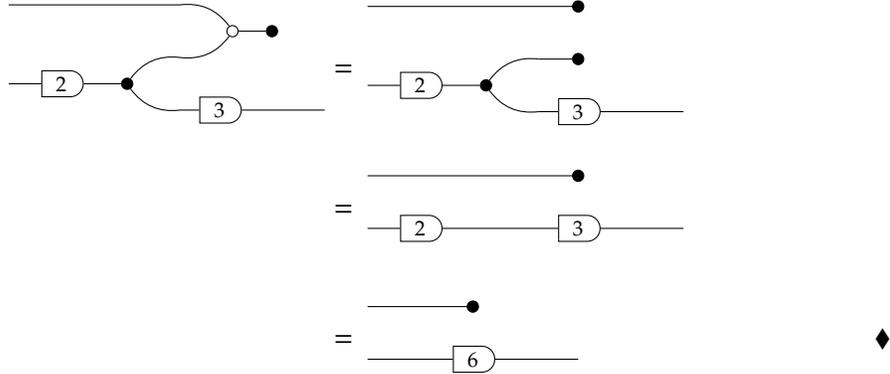
Example 5.48. For example, consider the signal flow graphs



They both represent the matrix

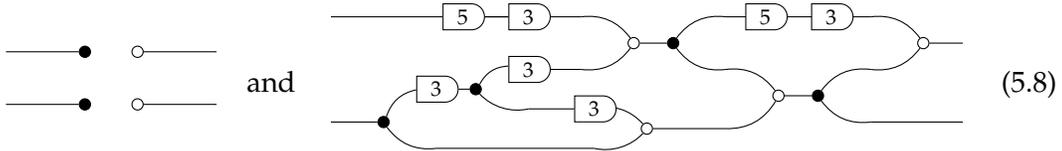
$$\begin{pmatrix} 0 & 6 \end{pmatrix}.$$

This means that one can be transformed into the other by using the above equations. Indeed, here



Exercise 5.49. For each matrix in Exercise 5.45, draw another signal flow graph that represents that matrix. Using the above equations and the prop axioms, prove that the two signal flow graphs represent the same matrix. \diamond

Exercise 5.50. Consider the signal flow graphs



1. Let $R = (\mathbb{N}, 0, +, 1, *)$. By examining the presentation of $\mathbf{Mat}(R)$ in Theorem 5.47, and without computing the matrices that they represent, prove that they do *not* represent the same matrix.
2. Now suppose the rig is $R = \mathbb{N}/3$; if you do not know what this means, just replace all 3's with 0's in the right-hand diagram of Eq. (5.8). Find what you would call a minimal representation of this diagram, using the presentation in Theorem 5.47. \diamond

5.4.2 Aside: monoid objects in a monoidal category

Various subsets of the equations in Theorem 5.47 encode structures that are familiar from many other parts of mathematics, e.g. representation theory. For example one can find the axioms for monoids, comonoids, Frobenius algebras, and (with a little rearranging) Hopf algebras sitting inside this collection. The first example, monoids, is particularly familiar to us by now, so we briefly discuss it below, both in algebraic terms (Definition 5.51) and in diagrammatic terms (Example 5.54).

Definition 5.51. A *monoid object* (M, μ, η) in a symmetric monoidal category (C, I, \otimes) is an object M of C together with morphisms $\mu: M \times M \rightarrow M$ and $\eta: I \rightarrow M$ such that

- (a) $(\mu \otimes \text{id}).\mu = (\text{id} \otimes \mu).\mu$ and
- (b) $(\text{id} \otimes \eta).\mu = \text{id} = (\eta \otimes \text{id}).\mu$.

A *commutative monoid object* is a monoid object that further obeys

- (c) $\sigma.\mu = \mu$.

where σ_M be the swap map on M in \mathcal{C} .

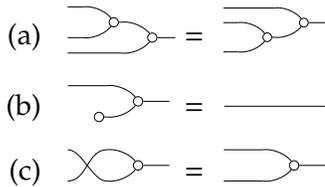
Monoid objects are so-named because they are an abstraction of the usual concept of monoid.

Example 5.52. A monoid is a monoid object in $(\mathbf{Set}, \times, 1)$. That is, it is a set M , a function $\mu: M \times M \rightarrow M$, which we denote by infix notation $*$, and an element $\eta \in M$, satisfying $(a * b) * c = a * (b * c)$ and $a * \eta = a = \eta * a$. \blacklozenge

Exercise 5.53. Consider the set \mathbb{R} of real numbers.

1. Show that if $\mu: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ is defined by $\mu(a, b) = a * b$ and if $\eta \in \mathbb{R}$ is defined to be $\eta = 1$, then $(\mathbb{R}, *, 1)$ satisfies the conditions of Definition 5.51.
2. Show that if $\mu: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ is defined by $\mu(a, b) = a + b$ and if $\eta \in \mathbb{R}$ is defined to be $\eta = 0$, then $(\mathbb{R}, +, 0)$ satisfies the conditions of Definition 5.51. \blacklozenge

Example 5.54. Graphically, we can depict $\mu = \succ$ and $\eta = \dashv$. Then axioms (a), (b), and (c) from Definition 5.51 become:



All three of these are found in Theorem 5.47. Thus we can immediately conclude the following: the triple $(1, \succ, \dashv)$ is a commutative monoid object in the prop $\mathbf{Mat}(R)$. \blacklozenge

Exercise 5.55. In Example 5.54, we said that the triple $(1, \succ, \dashv)$ is a commutative monoid object in the prop $\mathbf{Mat}(R)$. If $R = \mathbb{R}$ is the rig of real numbers, this means that we have a monoid structure on the set \mathbb{R} . But in Exercise 5.53 we gave two such monoid structures. Which one is it? \blacklozenge

Example 5.56. The triple $(1, \prec, \dashv)$ in $\mathbf{Mat}(R)$ forms a commutative monoid object in $\mathbf{Mat}(R)^{\text{op}}$. We hence also say that $(1, \prec, \dashv)$ forms a *co-commutative comonoid object* in $\mathbf{Mat}(R)$. \blacklozenge

Example 5.57. A *symmetric strict monoidal category*, is just a commutative monoid object in $(\mathbf{Cat}, \times, \mathbf{1})$. We will unpack this in Section 6.4.1. \blacklozenge

Example 5.58. A symmetric monoidal poset, which we defined in Definition 2.1, is just a commutative monoid object in $(\mathbf{Poset}, \times, \mathbf{1})$. \blacklozenge

Example 5.59. For those who know what tensor products of commutative monoids are (or can guess): A rig is a monoid object in the symmetric monoidal category $(\mathbf{CMon}, \otimes, \mathbb{N})$ of commutative monoids with tensor product. \blacklozenge

Exercise 5.60. Is there a symmetric monoidal structure on the category **Mon** of monoids and monoid homomorphisms for which a rig also a commutative monoid object in **Mon**? \diamond

Remark 5.61. If we present a prop \mathcal{M} generated by $\mu: 2 \rightarrow 1$ and $\eta: 0 \rightarrow 1$, with the three equations from Definition 5.51, we could call it “the theory of monoids”. This means that in any strict symmetric monoidal category \mathcal{C} , the monoid objects in \mathcal{C} correspond to functors $\mathcal{M} \rightarrow \mathcal{C}$. This sort of idea leads to the study of algebraic theories, due to Bill Lawvere and extended by many others; see Section 5.5.

5.4.3 Signal flow graphs: feedback and more

At this point in the story, we have seen that every signal flow graph represents a matrix, and this gives us a new way of reasoning about matrices. This is just the beginning of beautiful tale.

The pictorial nature of signal flow graphs invites us to play with them. While we normally draw the copy icon like so, \curvearrowright , we could just as easily reverse it and draw an icon \curvearrowleft . What might it mean? Let’s think again about the semantics of flow graphs.

The behavioral perspective A signal flow graph $g: m \rightarrow n$ takes an input $x \in R^m$ and gives an output $y \in R^n$. In fact, since this is all we care about, we might just think about representing a signal flow graph g as describing a set of pairs of inputs and outputs (x, y) . We’ll call this the *behavior* of g and denote it $B(g) \subseteq R^m \times R^n$. For example, the ‘copy’ flow graph



maps the input 1 to the output (1, 1), so we consider (1, (1, 1)) an element of its behavior. Similarly, so is $(x, (x, x))$ for every $x \in R$, thus we have

$$B(\curvearrowright) = \{(x, (x, x)) \mid x \in R\}.$$

In the abstract, the signal flow graph $g: m \rightarrow n$ has the behavior

$$B(g) = \{(x, S(g)(x)) \mid x \in R^m\} \subseteq R^m \times R^n. \tag{5.9}$$

Reversing icons This behavioral perspective clues us in on how to interpret the reversed diagrams discussed above. Reversing an icon $g: m \rightarrow n$ exchanges the inputs with the outputs, so if we denote this reversed icon by g^{op} , we must have $g^{\text{op}}: n \rightarrow m$. Thus if $B(g) \subseteq R^m \times R^n$ then we need $B(g^{\text{op}}) \subseteq R^n \times R^m$. One simple way to do this is to replace each (a, b) with (b, a) , so we would have

$$B(g^{\text{op}}) = \{(S(g)(x), x) \mid x \in R^m\} \subseteq R^n \times R^m. \tag{5.10}$$

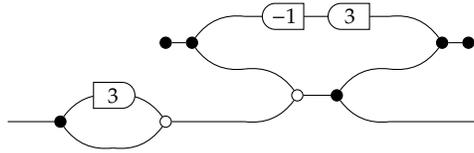
Exercise 5.62.

1. What is the behavior $B(\curvearrowleft)$ of the reversed addition icon $\curvearrowleft: 1 \rightarrow 2$?

2. What is the behavior $B(\curvearrowright)$ of the reversed copy icon, $\curvearrowright: 2 \rightarrow 1$? \diamond

Eqs. (5.9) and (5.10) give us formulas for interpreting signal flow graphs and their mirror images. But this would easily lead to disappointment, if we couldn't combine the two directions; luckily we can.

Combining directions What should the behavior of a diagram such as



be?

Let's formalize our thoughts a bit and begin by thinking about behaviors. The behavior of a signal flow graph $m \rightarrow n$ is a subset $B \subseteq R^m \times R^n$, i.e. a relation. Why not try to construct a prop where the morphisms $m \rightarrow n$ are relations?

We'll need to know how to compose two relations and take monoidal products. And if we want this prop of relations to contain the old prop $\mathbf{Mat}(R)$, we need the new compositions and monoidal products to generalize the old ones in $\mathbf{Mat}(R)$. Given signal flow graphs with matrices $M: m \rightarrow n$ and $N: n \rightarrow p$, we see that their behaviors are the relations $B_1 := \{(x, Mx) \mid x \in R^m\}$ and $B_2 := \{(y, Ny) \mid y \in R^n\}$, while the behavior of $M.N$ is the relation $\{(x, M.Nx) \mid x \in R^m\}$. Another way of writing this composite relation is

$$\{(x, z) \mid \text{there exists } y \in R^n \text{ such that } (x, y) \in B_1 \text{ and } (y, z) \in B_2\} \subseteq R^m \times R^p.$$

We shall use this as the general definition for composing two behaviors.

Definition 5.63. Let R be a rig. We define the prop \mathbf{Rel}_R of R -relations to have subsets $B \subseteq R^m \times R^n$ as morphisms. These are composed by the above composition rule, and we take the product of two sets to form their monoidal product.

Exercise 5.64. We went quickly through monoidal products $+$ in Definition 5.63. If $B \subseteq R^m \times R^n$ and $C \subseteq R^p \times R^q$ are morphisms in \mathbf{Rel}_R , write down $B + C$ in set-notation. \diamond

Full signal flow graphs Recall that above, e.g. in Definition 5.36, we wrote G_R for the set of generators of signal flow graphs. For each $g \in G_R$, we wrote g^{op} for its mirror image. So let's write $G_R^{\text{op}} := \{g^{\text{op}} \mid g \in G_R\}$ for the set of all the mirror images of generators. We define a prop

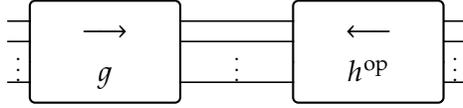
$$\mathbf{SFG}_R^+ := \mathbf{Free}(G_R \sqcup G_R^{\text{op}}). \tag{5.11}$$

We call a morphism in the prop \mathbf{SFG}_R^+ a (*non-simplified*) *signal flow graph*: these extend our simplified signal flow graphs from Definition 5.36 because now we can also use

the mirrored icons. By the universal property of free props, since we have said what the behavior of the generators is, we can define the behavior of any signal flow graph.

The following two exercises help us understand what this behavior is.

Exercise 5.65. Let $g: m \rightarrow n, h: \ell \rightarrow n$ be signal flow graphs. Note that $h^{\text{op}}: n \rightarrow \ell$ is a signal flow graph, and we can form the composite $g.h^{\text{op}}$:

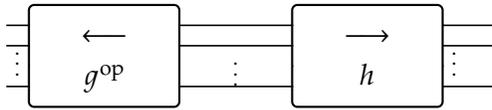


Show that the behavior of $g.h^{\text{op}}$ is equal to

$$\{(x, y) \mid S(g)(x) = S(h)(y)\} \subseteq R^m \times R^\ell. \tag{5.12}$$

◇

Exercise 5.66. Let $g: m \rightarrow n, h: m \rightarrow p$ be signal flow graphs. Note that $g^{\text{op}}: n \rightarrow m$ is a signal flow graph, and we can form the composite $g^{\text{op}}.h$



Show that the behavior of $g^{\text{op}}.h$ is equal to

$$\{(S(g)(x), S(h)(x)) \mid x \in R^m\}. \tag{5.13}$$

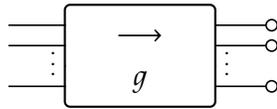
◇

Note the similarities and differences between this formula and Eq. (5.12).

Linear algebra via signal flow graphs

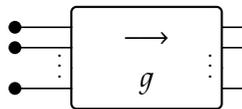
Exercise 5.67. Here is an exercise for those that know linear algebra. Let R be a field, and $g: m \rightarrow n$ a signal flow graph and let $S(g) \in \mathbf{Mat}(R)$ be the associated $(m \times n)$ -matrix (see Theorem 5.41).

1. Show that the composite of g with 0-reverses, shown here



is equal to the kernel of the matrix $S(g)$.

2. Show that the composite of discard-reverses with g , shown here



is equal to the image of the matrix $S(g)$.

3. Show that for any signal flow graph g , the subset $B(g) \subseteq R^m \times R^n$ is a linear subspace. That is, if $b_1, b_2 \in B(g)$ then so are $b_1 + b_2$ and $r * b_1$, for any $r \in R$. \diamond

In Exercise 5.67 we showed that the behavior of a signal flow graph is a linear relation, i.e. a relation whose elements can be added and multiplied by scalars $r \in R$. In fact the converse is true too: any linear relation $B \subseteq R^m \times R^n$ can be represented by a signal flow graph.

One can in fact give a complete presentation for linear relations on R , whose generating set is $G_R \sqcup G_R^{\text{op}}$ as in Eq. (5.11) and whose equations include those from Theorem 5.47 plus a few more. This presentation is sound and complete for linear relations, just like Theorem 5.47 gave us a sound and complete presentation for matrices. We will not discuss this, though see Section 5.5.

For now, we want to discuss where the “few more” equations, mentioned in the previous paragraph, come from. The answer is that linear relations—just like co-design problems in Chapter 4—form a compact closed category (see Definition 4.41). Our next goal is to describe this.

Compact closed structure Using the icons available to us for signal flow graphs, we can build morphisms that look like the ‘cup’ and ‘cap’:

$$\bullet \text{---} \bullet \text{---} \text{Cup} \quad \text{and} \quad \text{Cap} \text{---} \bullet \text{---} \bullet \tag{5.13}$$

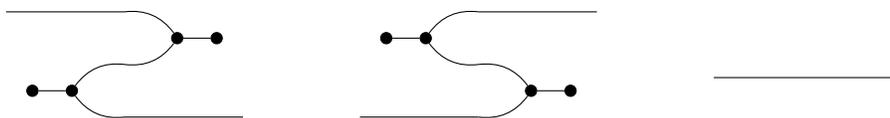
The behaviors of these graphs are respectively

$$\{(0, (x, x)) \mid x \in R\} \subseteq R^0 \times R^2 \quad \text{and} \quad \{((x, x), 0) \mid x \in R\} \subseteq R^2 \times R^0.$$

In fact, these work: the morphisms from Eq. (5.13) serve as the η_1 and ϵ_1 from Definition 4.41, i.e. the object 1 in the prop \mathbf{Rel}_R is dual to itself. Using monoidal products of these morphisms, one can show that any object in \mathbf{Rel}_R is dual to itself. This implies the following theorem.

Theorem 5.68. *The prop \mathbf{Rel}_R is a compact closed category in which every object $n \in \mathbb{N}$ is dual to itself, $n = n^*$.*

Graphically, this means that the three signal flow graphs



all represent the same relation.

To make our signal flow graphs simpler, we define new icons cup and cap by the equations

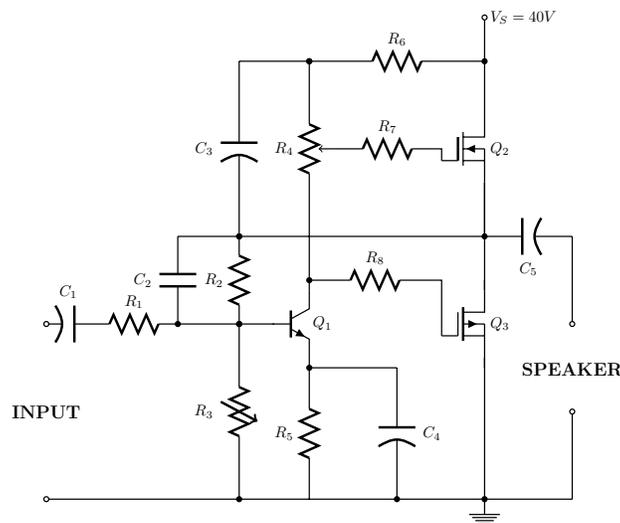
$$\text{Cup} := \bullet \text{---} \bullet \text{---} \text{Cup} \quad \text{and} \quad \text{Cap} := \text{Cap} \text{---} \bullet \text{---} \bullet$$

Perhaps the most significant idea in this chapter is the separation of structure into syntax and semantics, related by a functor. This is not only present in the running theme of studying signal flow graphs, but in our aside Section 5.4.2, where we talk, for example, about monoid objects in monoidal categories. The idea of functorial semantics is yet another due to Lawvere, first appearing in his thesis [Law04].

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, is an electric circuit that models an amplifier:



These diagrams have time-tested utility. In this chapter, we are interested in understanding the common mathematical structure that these models 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 resistors. When we understand the underlying structures that are shared whenever a model is well-represented by network diagrams, we can make comparisons between them 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—such as that for resource theories or co-design—every morphism has a domain and codomain. 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.

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, a lamp connected to a resistor:

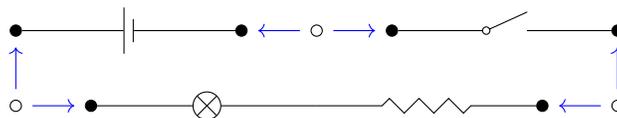


We want to connect them together. But there are many ways to connect them together. How should we describe the particular way that will form a light switch?

First, we claim that circuits should really be thought of as open circuits: they have the additional structure of certain locations, or ports, where we are allowed to connect them with other components. 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 \bullet . In this case it’s easy; the ports are simply the dangling end points of the wires.



Next, we have to describe which ports should be connected. We’ll do this by drawing empty circles \circ . These will be connection witnesses, saying ‘connect these two!’. This means there will be arrows from the circles to the ports.



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



But mathematics doesn't have a visual cortex with which to generate the same intuitions we have. Thus we need to specify formally how the new circuit with the marked points identified is actually constructed mathematically. As it turns out, we can do this with finite colimits in a given category \mathcal{C} .

But 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 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 structure.

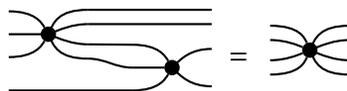
In summary, the first part of this chapter is devoted to the slogan 'colimits model connection'. In addition to universal constructions such as colimits, however, another way to describe interconnection is to use wiring diagrams. We go full circle when we find that these wiring diagrams are strongly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

Composition operations and wiring diagrams In this book we have seen the utility of defining syntactic or algebraic structures that describe under what sort composition operations make sense and can be performed in our application area. Examples include monoidal posets 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 $s_{m,n} : m \rightarrow n$. These icons, known as *spiders*,¹ are drawn as follows:



They obey rules encoding the idea that all that matters is which points are wired to which others using the spiders, and not the spiders themselves. This means we may fuse spiders. An example of such an equation is the following



A hypergraph category is one in whose wiring diagram iconography includes all such spiders for any given object (wire label), and thus whose morphisms we can compose in this network fashion.

¹Our spiders have any number of legs.

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** are the theory of hypergraph categories.

We can assemble all cospans in **FinSet** into an operad. Throughout this book we have talked about using free structures and presentations to create algebraic objects such as posets, categories, and props, tailored to the needs of a particular situation. Operads can be used to tailor the algebraic operations *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: Galois connections and adjunctions (Sections 1.5 and 3.4), limits (Section 3.5), and free and presented structures (Chapter 5). Here we turn our attention to colimits.

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 the same’ as elements in another set, and then join 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 join them together.

Definition 6.1. Let C be a category. An *initial object* in C is an object $\emptyset \in C$ such that for each object T in C there exists a unique morphism $t: \emptyset \rightarrow T$.

The symbol \emptyset is just a default name, a notation, intended to evoke the right idea; see Example 6.3 for the reason why we use the notation \emptyset , and Exercise 6.6 for a case when the default name \emptyset 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 poset is a bottom element—that is, an element that is less than every other element. ♦

Example 6.3. The initial object in **FinSet** is the empty set. Given any finite set T , there is a unique function $\emptyset \rightarrow T$, since \emptyset has no elements. ♦

Example 6.4. A category C need not have an initial object. For example, consider the category shown here:

$$C := \boxed{\begin{array}{ccc} A & \begin{array}{c} \xrightarrow{f} \\ \xrightarrow{g} \end{array} & B \\ \bullet & & \bullet \end{array}}$$

If there were to be an initial object \emptyset , it would either be $\emptyset = A$ or $\emptyset = B$. Either way, we need to show that for each object $T \in \text{Ob}(C)$ (i.e. for both $T = A$ and $T = B$) there is a unique morphism $\emptyset \rightarrow T$. Trying the case $\emptyset = A$ this condition fails when $T = B$: there are two morphisms $A \rightarrow B$, not one. And trying the case $\emptyset = B$ this condition fails when $T = A$: there are no morphisms $B \rightarrow A$, not one. \blacklozenge

Exercise 6.5. For each of the graphs below, consider the free category on that graph, and say whether it has an initial object.

1. $\boxed{\begin{array}{c} a \\ \bullet \end{array}}$ 2. $\boxed{\begin{array}{ccccc} a & & b & & c \\ \bullet & \rightarrow & \bullet & \rightarrow & \bullet \end{array}}$ 3. $\boxed{\begin{array}{cc} a & b \\ \bullet & \bullet \end{array}}$ 4. $\boxed{\begin{array}{c} a \\ \bullet \curvearrowright \end{array}}$ \blacklozenge

Exercise 6.6. 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 $f: R \rightarrow S$ 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? \blacklozenge

Exercise 6.7. 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 universal? What is the “comparable object”? \blacklozenge

Remark 6.8. As mentioned in Remark 3.69, 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.

6.2.2 Coproducts

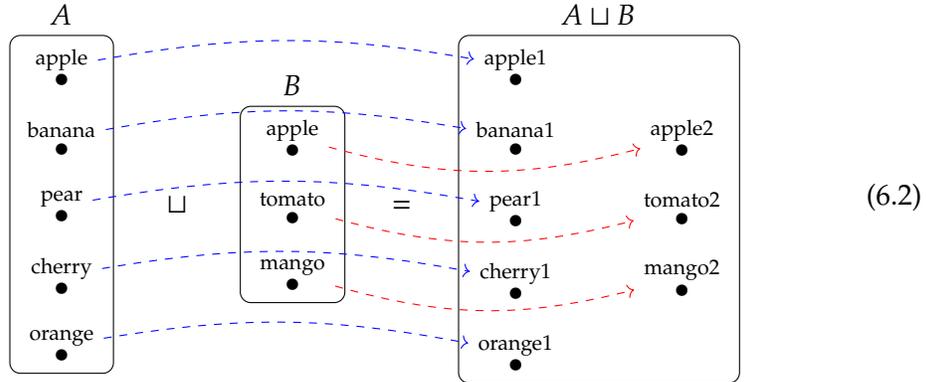
Coproducts generalize both joins in a poset and disjoint unions of sets.

Definition 6.9. Let A and B be objects in a category C . A *coproduct* of A and B is an object denoted $A + B$, together with a pair morphisms $(\iota_A: A \rightarrow A + B, \iota_B: B \rightarrow A + B)$ such that for all objects T and pairs of maps $(f: A \rightarrow T, g: B \rightarrow T)$ there exists a unique map $[f, g]: A + B \rightarrow T$ such that the following diagram commutes:

$$\begin{array}{ccccc} A & \xrightarrow{\iota_A} & A + B & \xleftarrow{\iota_B} & B \\ & \searrow f & \downarrow [f, g] & \swarrow g & \\ & & T & & \end{array} \tag{6.1}$$

Exercise 6.10. Explain why coproducts in a poset are the same as joins. ◇

Example 6.11. 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 set $A \sqcup B$ together with the inclusion functions $\iota_A: A \rightarrow A \sqcup B$ and $\iota_B: B \rightarrow A \sqcup B$.



Suppose we have functions $f: A \rightarrow T$ and $g: B \rightarrow T$ for some other set T , unpictured. The universal property of coproducts says there is a unique function $[f, g]: A \sqcup B \rightarrow T$ such that $\iota_A.[f, g] = f$ and $\iota_B.[f, g] = g$. What is it? If $a \in A$, we see from the condition Eq. (6.1) that we must have

$$[f, g](\iota_A(a)) = f(a),$$

and the analogous statement is true for all $b \in B$, so we have:

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

Exercise 6.12. 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. (6.2). Consider the function $f: A \rightarrow T$ sending each element of A to the first letter of its label, e.g. $f(\text{apple}) = a$. Let $g: B \rightarrow T$ 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 nine elements of $A \sqcup B$. ◇

Exercise 6.13. Suppose a category C has coproducts, denoted $+$, and an initial object, denoted \emptyset . Then we can show $(C, +, \emptyset)$ is a symmetric monoidal category (recall Definition 4.32). In this exercise we develop the data.

1. Show that $+$ extends to a functor $C \times C \rightarrow C$. In particular, how does it act on morphisms in $C \times C$?
2. Using the relevant universal properties, show that there are isomorphisms
 - a) $A + \emptyset \rightarrow A$
 - b) $\emptyset + A \rightarrow A$.
 - c) $(A + B) + C \rightarrow A + (B + C)$.
 - d) $A + B \rightarrow B + A$.

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 6.20.

Definition 6.14. Let C be a category and let $f: A \rightarrow X$ and $g: A \rightarrow Y$ be morphisms in C that have a common domain. The *pushout* $X +_A Y$ is the colimit of the diagram

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

In more detail, a pushout consists of an object $X +_A Y$ and morphisms $\iota_X: X \rightarrow X +_A Y$ and $\iota_Y: Y \rightarrow X +_A Y$ satisfying 1.) and 2.) below.

1.) The diagram

$$\begin{array}{ccc} A & \xrightarrow{f} & X \\ g \downarrow & \lrcorner & \downarrow \iota_X \\ Y & \xrightarrow{\iota_Y} & X +_A Y \end{array} \tag{6.3}$$

commutes. (We will explain the “ \lrcorner ” symbol below.)

2.) For all objects T and morphisms $x: X \rightarrow T$, $y: Y \rightarrow T$, if the diagram

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

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

$$\begin{array}{ccc} A & \xrightarrow{f} & X \\ g \downarrow & & \downarrow \iota_X \\ Y & \xrightarrow{\iota_Y} & X +_A Y \end{array} \begin{array}{c} \searrow x \\ \downarrow \\ \xrightarrow{y} \\ \downarrow \\ \xrightarrow{t} \end{array} T \tag{6.4}$$

commutes.

If $X +_A Y$ is a pushout, we denote that fact by drawing the commutative square Eq. (6.3), together with the \lrcorner symbol as shown; we call this a *pushout square*.

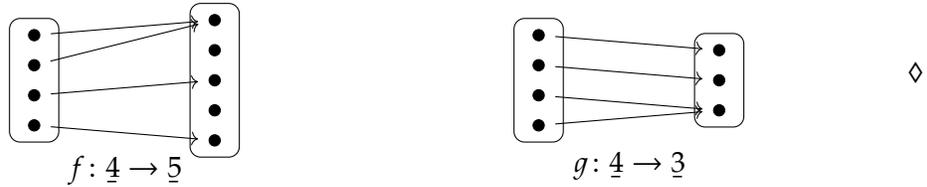
We further call ι_X the *pushout of g along f* , and similarly ι_Y the *pushout of f along g* .

Example 6.15. Pushouts in a poset are the same as coproducts: they are both joins. \blacklozenge

Example 6.16. In the category **FinSet**, pushouts always exist. The pushout of functions $f: A \rightarrow X$ and $g: A \rightarrow Y$ 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 $x \sim y$ if there exists a in A such that $f(a) = x$ and $g(a) = y$.

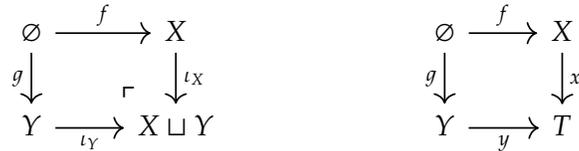
We can think of this in terms of interconnection too. Each element of a provides a connection between $f(a)$ in X and $g(a)$ in Y . The pushout is the set of connected components of $X \sqcup Y$. ♦

Exercise 6.17. What is the pushout of the functions $f: \underline{4} \rightarrow \underline{5}$ and $g: \underline{4} \rightarrow \underline{3}$ pictured below?



Example 6.18. Suppose a category C has an initial object \emptyset . For any two objects $X, Y \in \text{Ob } C$, there is a unique morphism $f: \emptyset \rightarrow X$ and a unique morphism $g: \emptyset \rightarrow Y$; this is what it means for \emptyset to be initial.

The diagram $X \xleftarrow{f} \emptyset \xrightarrow{g} Y$ has a pushout in C iff X and Y have a coproduct in C , and they are the same. Indeed, suppose X and Y have a coproduct $X \sqcup Y$; then the diagram to the left



commutes (why?¹), and for any object T and commutative diagram as to the right, there is a unique map $X \sqcup Y \rightarrow T$ making the diagram as in Eq. (6.4) commute (why?²). This shows that $X \sqcup Y$ is a pushout, $X \sqcup_{\emptyset} Y = X \sqcup Y$.

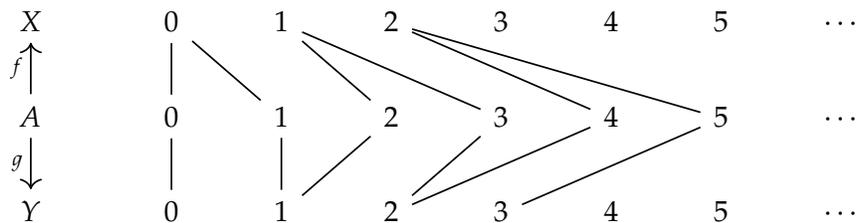
Similarly, if a pushout $X \sqcup_{\emptyset} Y$ exists, then it satisfies the universal property of the coproduct (why?³). ♦

Exercise 6.19. In Example 6.18 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 6.20. Let $A = X = Y = \mathbb{N}$. Consider the functions $f: A \rightarrow X$ and $g: A \rightarrow Y$ given by the ‘floor’ functions, $f(a) := \lfloor a/2 \rfloor$ and $g(a) := \lfloor (a+1)/2 \rfloor$.



If T is any other set and we have maps $x: X \rightarrow T$ and $y: Y \rightarrow T$ that commute with f and g , then this commutativity implies that

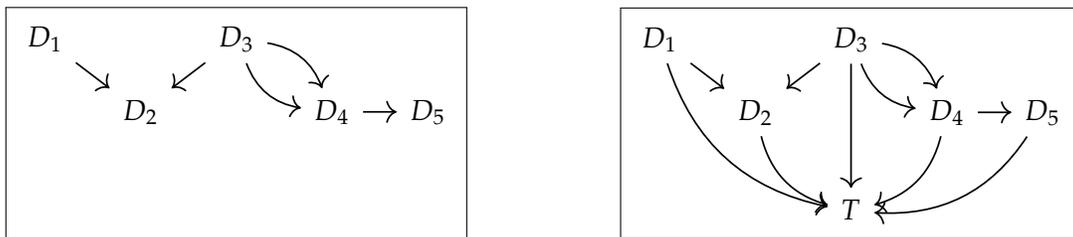
$$y(0) = y(g(0)) = x(f(0)) = x(0).$$

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. One can keep repeating this and find that every element of Y and every element of X go to t ! One can use the above sort of argument, along with mathematical induction,² to 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 C is a terminal object in a category of cones over a diagram $D: \mathcal{J} \rightarrow C$, a colimit is an initial object in a category of cocones over some diagram $D: \mathcal{J} \rightarrow C$. 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.³

In Definition 3.85, cocones in C are defined to be cones in C^{op} . For visualization purposes, if $D: \mathcal{J} \rightarrow C$ looks like the diagram to the left, then a cone on it shown in the diagram to the right:



Here, any two parallel paths that end at T are considered the same.

Definition 6.21. We say that a category C has finite colimits if a colimit, $\text{colim}_{\mathcal{J}} D$, exists whenever \mathcal{J} is a finite category and $D: \mathcal{J} \rightarrow C$ is a diagram.

Example 6.22. A seemingly tautological example to check is that initial objects are finite colimits.

The initial object in a category C , if it exists, is the colimit of the functor $!: \mathbf{0} \rightarrow C$, where $\mathbf{0}$ is the category with no objects and no morphisms, and $!$ is the unique such

²We do not discuss mathematical induction in this book, but it is a very important proof technique based on a sort of infinite-domino effect: knock down the first, each knocks down the next, and they all fall. This can be formulated in terms of a universal property held by the set \mathbb{N} of natural numbers, its “first” element 0 and its “next” function $\mathbb{N} \rightarrow \mathbb{N}$.

³If a category \mathcal{J} has finitely many morphisms, then it must have finitely many objects, because each object $j \in \text{Ob } \mathcal{J}$ has its own identity morphism id_j . In this case we say that \mathcal{J} is a *finite category*.

functor. Indeed, a cocone over $!$ is just an object of C , and so the initial cocone over $!$ is just the initial object of C .

Note that $\mathbf{0}$ has finitely many objects; thus initial objects are finite colimits. \blacklozenge

To check that a category has *all* finite colimits, it's enough to check it has some simpler forms of colimit.

Proposition 6.23. *A category has finite colimits if and only if it has an initial object and pushouts.*

Proof. We will not give precise details here, but the key idea is an inductive one: every finite diagram can be built using the empty diagram and pushout diagrams. Full details can be found in [Mac98]. \square

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

Corollary 6.24. *The categories **FinSet** and **Set** have finite colimits.*

In Theorem 3.79 we gave a general formula for computing finite limits in **Set**. It is also possible to give a formula for finite colimits. Rather than take a subset of the product of all the sets in the diagram, we instead take a quotient of the disjoint union of all the sets in the diagram.

Theorem 6.25. *Let \mathcal{J} be presented by the finite graph (V, A, s, t) and some equations, and let $D: \mathcal{J} \rightarrow \mathbf{Set}$ be a diagram. Consider the set*

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

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 $a: v \rightarrow w$ in \mathcal{J} such that $D(a)(d) = e$. Then this set, together with the functions $\iota_v: D(v) \rightarrow \operatorname{colim}_{\mathcal{J}} D$ given by sending $d \in D(v)$ to its equivalence class, constitutes a colimit of D .

Example 6.26. 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 \emptyset : there are no $v \in V$. \blacklozenge

Example 6.27. A coproduct is a colimit on the graph $\mathcal{J} = \begin{array}{|c|c|} \hline v_1 & v_2 \\ \hline \bullet & \bullet \\ \hline \end{array}$. A functor $D: \mathcal{J} \rightarrow \mathbf{Set}$ 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 6.25 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. \blacklozenge

Example 6.28. If \mathcal{J} is the category $\mathbf{1} = \boxed{\bullet}$, the formula in Theorem 6.25 yields the set

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

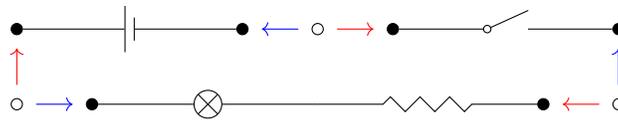
This is isomorphic to the set $D(v)$. In other words, if X is a set considered as a diagram $X: \mathbf{1} \rightarrow \mathbf{Set}$, then its colimit (like its limit) is just X again. \blacklozenge

Exercise 6.29. Use the formula to show that pushouts—colimits on a diagram $X \xleftarrow{f} N \xrightarrow{g} Y$ —agree with the description we gave in Example 6.16. \blacklozenge

Example 6.30. 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 $X \begin{matrix} \xrightarrow{f} \\ \xrightarrow{g} \end{matrix} Y$ on this graph in \mathbf{Set} . The coequalizer of this diagram is the set of equivalence classes of Y under equivalence relation generated by $y \sim y'$ 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 \circ , and Y be the set of black circles \bullet , the blue and red arrows respectively define functions $f, g: X \rightarrow Y$. Let's leave the actual circuit components out of the picture for now; we're just interested in the dots. What is the coequalizer?

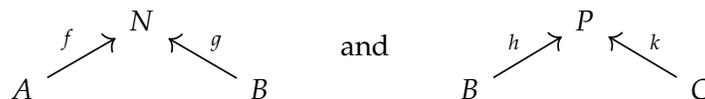
It is a three element set, consisting of one element for each newly-connected pair of \bullet 's. Thus the colimit describes the set of terminals after performing the interconnection operation. In Section 6.4 we'll see how to keep track of the circuit components too. \blacklozenge

6.2.5 Cospans

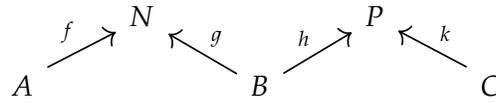
When a category \mathcal{C} has finite colimits, an extremely useful way to package them is by considering the category of cospans in \mathcal{C} .

Definition 6.31. Let \mathcal{C} be a category. A *cospan* in \mathcal{C} is just a pair of morphisms to a common object $A \rightarrow N \leftarrow B$. The common object N is called the *apex* of the cospan and the other two objects A and B are called its *feet*.

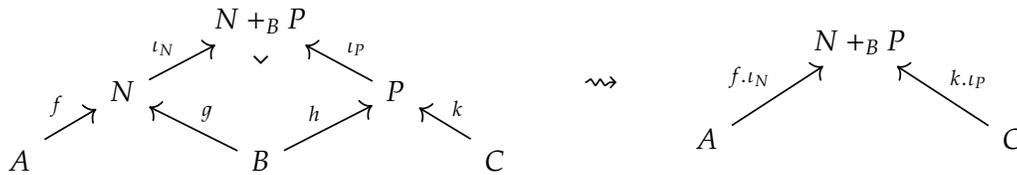
If we want to say that cospans form a category, we should begin by saying how composition would work. So suppose we have two cospans in \mathcal{C}



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 C , as shown on the left, we can extract a new cospan in C , as shown on the right:



It might look like we have achieved our goal, but we're missing some things. First, we need an identity on every object $C \in \text{Ob } C$; but that's not hard: use $C \rightarrow C \leftarrow C$ where both maps are identities in C . More importantly, we don't know that C has all pushouts, so we don't know that every two sequential morphisms $A \rightarrow B \rightarrow C$ 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 $A \xrightarrow{f} P \xleftarrow{g} B$ and $A' \xrightarrow{f'} P' \xleftarrow{g'} B'$ equivalent if there exists an isomorphism $p: P \rightarrow P'$ such that $f.p = f'$ and $g.p = g'$.

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

Recall from Proposition 6.23 that a category C has all finite colimits iff it has an initial object and all pushouts.

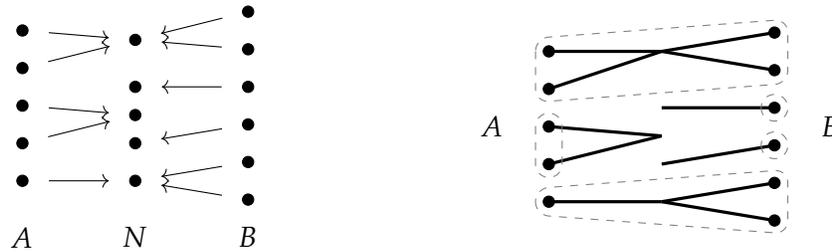
Definition 6.32. Let C be a category with finite colimits. Then there exists a category \mathbf{Cospan}_C with the same objects as C , i.e. $\text{Ob}(\mathbf{Cospan}_C) = \text{Ob}(C)$, where the morphisms $A \rightarrow B$ are the (equivalence classes of) cospans from $A \rightarrow B$, and composition is given by the above pushout construction.

There is a symmetric monoidal structure on this category, denoted $(\mathbf{Cospan}_C, \emptyset, +)$. The monoidal unit is the initial object \emptyset and the monoidal product is given by coproduct. The coherence isomorphisms, e.g. $A + \emptyset \cong A$, can be defined in a similar way to those in Exercise 6.13.

It takes some work to verify that $(\mathbf{Cospan}_C, \emptyset, +)$ from Definition 6.32 really does satisfy all the axioms of a symmetric monoidal category, but it does; see e.g. [Fon16].

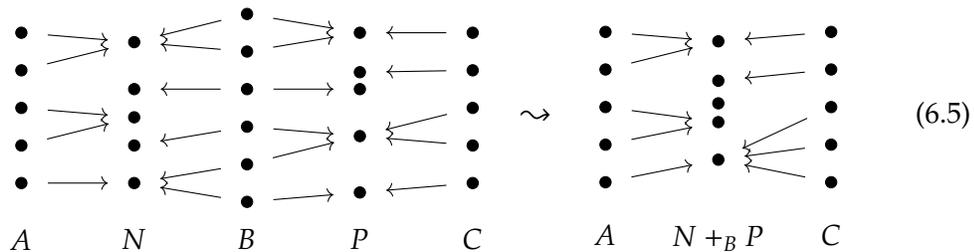
Example 6.33. The category **FinSet** has finite colimits (see 6.24). So, we can define a symmetric monoidal category $\mathbf{Cospan}_{\mathbf{FinSet}}$. What does it look like? It looks a lot like wires connecting ports.

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 a to $f(a)$ and 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. We also draw a dotted line around points that are connected, to emphasize an important perspective, that cospans allow us to mark certain ports as 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 6.32. Here is an example of cospan composition, where all the functions are depicted with arrow notation:

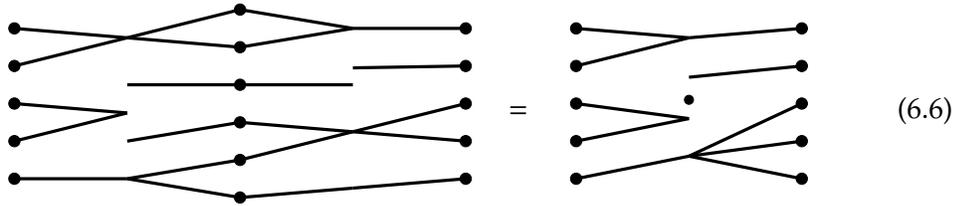


The monoidal product is given by just by the disjoint union of two cospans; in pictures it is simply combining two cospans by placing one above another. \blacklozenge

Exercise 6.34. In Eq. (6.5) we showed morphisms $A \rightarrow B$ and $B \rightarrow C$ in $\mathbf{Cospan}_{\mathbf{FinSet}}$. Draw their monoidal product as a morphism $A + B \rightarrow B + C$ in $\mathbf{Cospan}_{\mathbf{FinSet}}$. \blacklozenge

Exercise 6.35. Depicting the composite of cospans in Eq. (6.5) with the wire notation

gives



Comparing Eq. (6.5) and Eq. (6.6), describe the composition rule in $\mathbf{Cospan}_{\mathbf{FinSet}}$ in terms of wires and connected components. \diamond

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. But to define hypergraph categories, it is useful to first introduce Frobenius monoids.

6.3.1 Special commutative Frobenius monoids

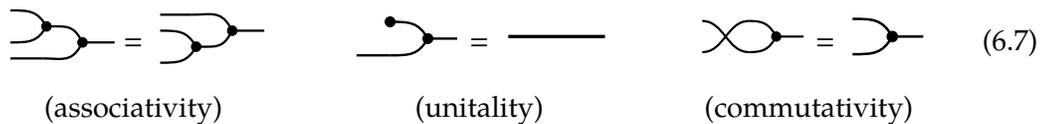
The pictures of cospans we saw above, e.g. in Eq. (6.6) look something like icons in signal flow graphs (see Section 5.3.2): various wires merge and split, initialize and terminate. And these follow the same rules they did for linear relations, which we briefly discussed in Definition 5.63. There’s a lot of potential for confusion, so let’s start from scratch and build back up.

In any symmetric monoidal category (C, I, \otimes) , recall that objects can be drawn as wires and morphisms can be drawn as boxes. Particularly note-worthy 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 5.51 that a commutative monoid (X, μ, η) in symmetric monoidal category (C, I, \otimes) is an object X of C together with (noteworthy) morphisms



obeying

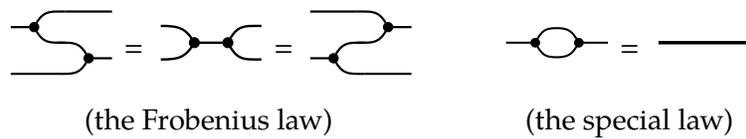


where \times is the symmetry on $X \otimes X$. A cocommutative comonoid (X, δ, ϵ) is an object X with maps $\delta: X \rightarrow X \otimes X, \epsilon: X \rightarrow I$, obey the mirror image laws to those above.

Suppose X has both the structure of a commutative monoid and cocommutative comonoid, and consider a wiring diagram built only from the icons $\mu, \eta, \delta,$ and $\epsilon,$ 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.

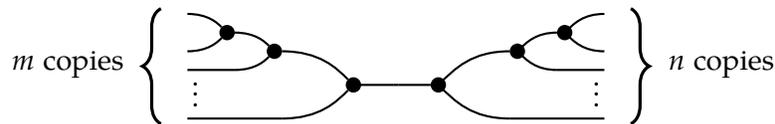
If you want to go all the way and say “all I care about is which port is connected to which; I don't even care about left and right”, then you need a few more axioms to say how the morphisms μ and $\delta,$ the splitter and the merger, interact.

Definition 6.36. A *special commutative Frobenius monoid* $(X, \mu, \eta, \delta, \epsilon)$ in a symmetric monoidal category (C, I, \otimes) consists of a commutative monoid (X, μ, η) and a cocommutative comonoid (X, δ, ϵ) that satisfy the six equations above ((co-)associativity, (co-)unitality, (co-)commutativity) and that further obey the following three equations:



With these two equations, it turns out that two morphisms $X^{\otimes m} \rightarrow X^{\otimes n}$ —defined by composing and tensoring identities on X and the noteworthy morphisms $\mu, \delta,$ 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. We denote id_X by X for readability reasons.

Definition 6.37. Let $(X, \mu, \eta, \delta, \epsilon)$ be a special commutative Frobenius monoid in a monoidal category (C, I, \otimes) . Let $m, n \in \mathbb{N}$. Define $s_{m,n}: X^{\otimes m} \rightarrow X^{\otimes n}$ 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 6.38. *Let $(X, \mu, \eta, \delta, \epsilon)$ be a special commutative Frobenius monoid in a monoidal category (C, I, \otimes) . Suppose that we have a map $f: X^{\otimes m} \rightarrow X^{\otimes n}$ each constructed from spiders and the symmetry map $\sigma: X^{\otimes 2} \rightarrow X^{\otimes 2}$ using composition and the monoidal product, and such that the string diagram of f has only one connected component. Then $f = s_{m,n}$.*

Example 6.39. 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 6.38 immediately implies they are equal:



Exercise 6.40. Which morphisms in the following list are 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 5.24.

Theorem 6.41. *Write $G = \{\mu, \eta, \delta, \epsilon\}$ and define $in, out: G \rightarrow \mathbb{N}$ as follows: $in(\mu) = 2, out(\mu) = 1, in(\eta) = 0, out(\eta) = 1, in(\delta) = 1, out(\delta) = 2, in(\epsilon) = 1, out(\epsilon) = 0$. Let E be the set of special commutative Frobenius monoid axioms, the nine equations from Definition 6.36. Then the free prop on (G, E) is equivalent, as a symmetric monoidal category, to $\mathbf{Cospans}_{\mathbf{FinSet}}$.*

We will not explain precisely what it means to be equivalent as a symmetric monoidal category, but instead just comment that the idea is similar to that of equivalence of categories, as explained in Remark 3.49.

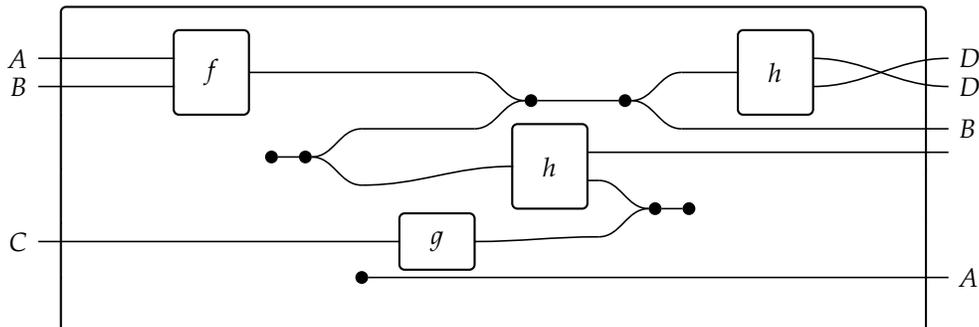
Thus we see that ideal wires, connectivity, cospans, and Frobenius monoids are all intimately related. We use Frobenius monoids as a way to define a new categorical structure, which captures the grammar of circuit diagrams.

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 labelled 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 specified ones.

Exercise 6.42.

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? \diamond

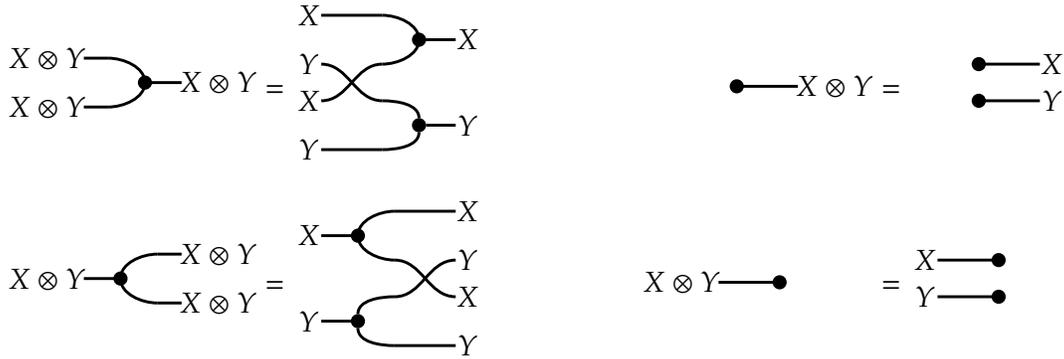
Thus hypergraph categories are general enough to talk about all network-style diagrammatic languages, like circuit diagrams.

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 6.43. A *hypergraph category* is a symmetric monoidal category (C, I, \otimes) in which each object X is equipped with a special commutative Frobenius structure $(X, \mu_X, \delta_X, \eta_X, \epsilon_X)$ such that



for all objects X, Y .

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

Example 6.44. For any C with finite colimits, \mathbf{Cospan}_C is a hypergraph category. The morphisms $\mu_X, \delta_X, \eta_X, \epsilon_X$ for each object X are constructed using the maps defined by the universal properties of colimits:

$$\begin{aligned} \mu_X &:= (X + X \xrightarrow{[\text{id}_X, \text{id}_X]} X \xleftarrow{\text{id}_X} X) \\ \eta_X &:= (\emptyset \xrightarrow{!} X \xleftarrow{\text{id}_X} X) \\ \delta_X &:= (X \xrightarrow{\text{id}_X} X \xleftarrow{[\text{id}_X, \text{id}_X]} X + X) \\ \epsilon_X &:= (X \xrightarrow{\text{id}_X} X \xleftarrow{!} \emptyset) \end{aligned} \quad \blacklozenge$$

Exercise 6.45. By Example 6.44, the category $\mathbf{Cospan}_{\mathbf{FinSet}}$ is a hypergraph category. (In fact it is a hypergraph prop.) Draw the Frobenius maps in $\mathbf{Cospan}_{\mathbf{FinSet}}$ using both the function and wiring depictions as in Example 6.33 \blacklozenge

Exercise 6.46. Using your knowledge of colimits, show that the maps defined in Example 6.44 do indeed obey the Frobenius law, the unitality law, and the special law; see Eq. (6.7) and Definition 6.36. \blacklozenge

Exercise 6.47. Recall the monoidal category $(\mathbf{Corel}, \emptyset, \sqcup)$ (Example 4.43), whose objects are finite sets and whose morphisms are corelations. Given a finite set X , define the corelation $\mu_X: X \sqcup X \rightarrow \sqcup X$ 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 $\delta_X: X \rightarrow X \sqcup X$ in the same way, and define $\eta_X: \emptyset \rightarrow X$ and $\epsilon_X: X \rightarrow \emptyset$ such that no two elements of $X = \emptyset \sqcup X = X \sqcup \emptyset$ are equivalent.

Using the corelation diagrams of Example 4.43, show that these maps define a special commutative Frobenius monoid $(X, mu_X, \eta_X, \delta_X, \epsilon_X)$. Conclude that **Corel** is a hypergraph category. \diamond

Example 6.48. The prop of linear relations, which we briefly mentioned in Exercise 5.67, 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. \diamond

We can generalize the construction we gave in Theorem 5.68.

Proposition 6.49. *Hypergraph categories are self-dual compact closed categories, if we define the cup and cap to be*

$$\text{cup} := \bullet \bullet \text{ and } \text{cap} := \text{cup}$$

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

$$\begin{aligned} \text{cup} &= \text{cup} && \text{(definition)} \\ &= \text{cup} && \text{(Frobenius)} \\ &= \text{cup} && \text{(unitality)} \end{aligned}$$

Exercise 6.50!

\square

Exercise 6.50. Fill in the missing diagram in the proof of Proposition 6.49. \diamond

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, these are known as symmetric monoidal functors. Then we introduce a general method—that of decorated cospans—for producing hypergraph categories. This will tie up lots of loose ends: colimits, cospans, circuits, and hypergraph categories.

6.4.1 Symmetric monoidal functors

Rough Definition 6.51. Let (C, I_C, \otimes_C) and (D, I_D, \otimes_D) be symmetric monoidal categories. To specify a *symmetric monoidal functor* (F, φ) between them,

- (i) one specifies a functor $F: C \rightarrow D$;
- (ii) one specifies a morphism $\varphi_I: I_D \rightarrow F(I_C)$.
- (iii) for each $c_1, c_2 \in \text{Ob}(C)$, one specifies a morphism

$$\varphi_{c_1, c_2}: F(c_1) \otimes_D F(c_2) \rightarrow F(c_1 \otimes_C c_2),$$

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 C and D .

Example 6.52. Consider the power set functor $P: \mathbf{Set} \rightarrow \mathbf{Set}$. This sends a set S to its set of subsets $P(S) := \{R \subseteq S\}$, and sends a function $f: S \rightarrow T$ to the image map $\text{im}_f: P(S) \rightarrow P(T)$, 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 4.35. To make P a symmetric monoidal functor, we need to specify a function $\varphi_I: \{1\} \rightarrow P(\{1\})$ and for all sets S and T , a functor $\varphi_{S,T}: P(S) \times P(T) \rightarrow P(S \times T)$. These functions are straightforward to define. Indeed, we simply define $\varphi_I(1)$ to be the set $\{1\} \in P(\{1\})$, and given subsets $A \subseteq S$ and $B \subseteq T$, we define $\varphi_{S,T}(A, B) := A \times B \subseteq S \times T$. \blacklozenge

Exercise 6.53. Check that the maps $\varphi_{S,T}$ defined in Example 6.52 are natural in S and T . In other words, given $f: S \rightarrow S'$ and $g: T \rightarrow T'$, show that the diagram below commutes:

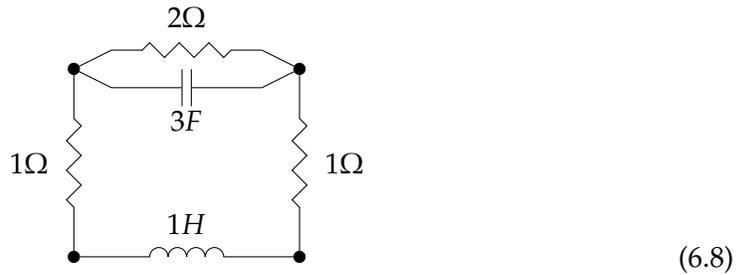
$$\begin{array}{ccc} P(S) \times P(T) & \xrightarrow{\varphi_{S,T}} & P(S \times T) \\ \downarrow [\text{im}_f, \text{im}_g] & & \downarrow \text{im}_{f \times g} \\ P(S') \times P(T') & \xrightarrow{\varphi_{S',T'}} & P(S' \times T') \end{array} \quad \blacklozenge$$

6.4.2 Decorated cospans

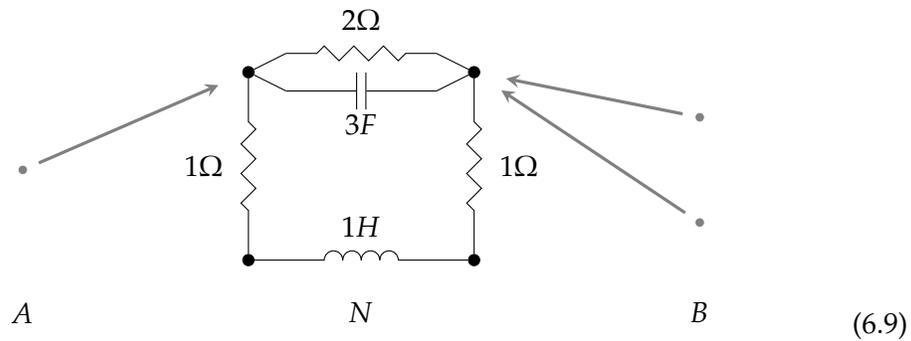
Now that we have briefly introduced symmetric monoidal functors, we return to the task at hand: constructing a hypergraph category of circuits. To do so, we introduce the method of decorated cospans.

Circuits have lots of internal structure, but they also have some external ports—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 circuit:

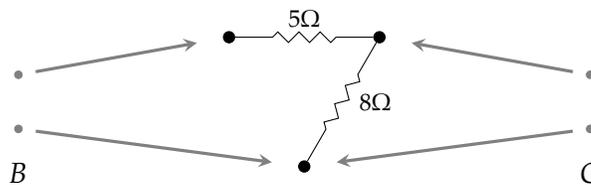


We might formally consider this as a graph on the set of four ports, where each edge is labelled by a type of circuit component (for example, the top edge would be labelled as a resistor of resistance 2Ω). For this circuit to be a morphism in some category, i.e. in order to allow for interconnection,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



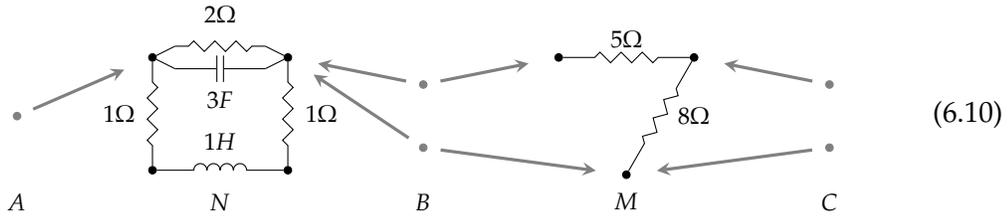
Let N be the set of nodes of the circuit. Here the finite sets A , B , and N are sets consisting of one, two, and four elements respectively, drawn as points, and the values of the functions $A \rightarrow N$ and $B \rightarrow N$ are indicated by the grey arrows. This forms a cospan in the category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

Suppose given another such decorated cospan with input B

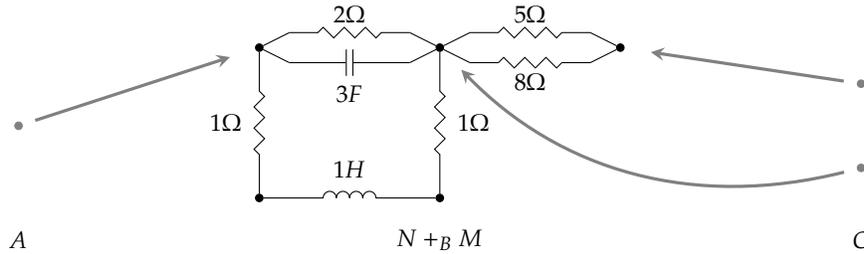


Since the output of the first equals the input of the second (both are B), we can stick

them together into a single diagram:



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



We’ve seen this sort of gluing before when we defined composition of cospans in Definition 6.32. But now there’s this whole “decoration” thing; our goal is to formalize it.

Definition 6.54. Let C be a category with finite colimits, and $(F, \varphi): (C, +) \rightarrow (\mathbf{Set}, \times)$ be a symmetric monoidal functor. An F -decorated cospan is a pair consisting of a cospan $A \xrightarrow{i} N \xleftarrow{o} B$ in C together with an element $s \in F(N)$.⁴ We call (F, φ) the *decoration functor* and s the *decoration*.

The intuition here is that, for each object $N \in C$, 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 decorations of N nodes that F has available.

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 $B \xrightarrow{h} P \xleftarrow{k} C$, paired with the element $F([\iota_N, \iota_P])(\varphi_{N,P}(s, t)) \in F(N +_B P)$.

⁴Just like in Definition 6.32, 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.

Don't worry if that's too abstract and you don't feel like chasing down what it all means. We'll continue to work through it in a concrete case, and leave the formalism here in case you want to look at it later. Let's record a theorem and then move on to circuits, for real this time.

Theorem 6.55. *Given a category C with finite colimits and a symmetric monoidal functor $(F, \varphi): (C, +) \rightarrow (\mathbf{Set}, \times)$, there is a hypergraph category \mathbf{Cospan}_F with objects the objects of C , and morphisms equivalence classes of F -decorated cospans.*

The symmetric monoidal and hypergraph structures are derived from those on \mathbf{Cospan}_C .

Exercise 6.56. Suppose you're worried that the notation \mathbf{Cospan}_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 $F(c) := \{*\}$." What do they mean? \diamond

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.

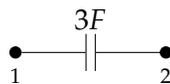
Let's build some circuits To begin, we'll need to choose which components we want in our circuit. This is simply a matter of what's in our toolbox. For now, we'll just consider passive linear circuits. That is, define a set

$$C := \{x\Omega \mid x \in \mathbb{R}^+\} \sqcup \{xF \mid x \in \mathbb{R}^+\} \sqcup \{xH \mid x \in \mathbb{R}^+\}.$$

To be clear, the Ω , F , and H are just labels; the above set is isomorphic to $\mathbb{R}^+ \sqcup \mathbb{R}^+ \sqcup \mathbb{R}^+$. But we write C this way to remind us that it consists of circuit components: the set of all possible resistances, capacitances, and inductances. If we wanted, we could also add lights and batteries and switches, 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 (N, A, s, t) , where $s, t: A \rightarrow N$ are the source and target functions, together with a function $\ell: A \rightarrow C$ labeling each edge with a certain circuit component from C .

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



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 capacitor could also be represented by the data $N = \{1, 2\}$, $A = \{e\}$, $s(e) = 2$, $t(e) = 1$,

and $\ell(e) = 3F$. This may not seem the most obvious or concise choice, but it is convenient for some ways of working with circuits, just as in linear algebra it is often convenient to represent a vector space via a basis.

Exercise 6.57. Write a tuple (N, A, s, t, ℓ) that represents the circuit in Eq. (6.8). \diamond

If we want C -circuits to be our decorations, let's use them to define a decoration functor (F, φ) as in Definition 6.54. We start by defining the functor part

$$F: (\mathbf{FinSet}, +) \longrightarrow (\mathbf{Set}, \times)$$

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

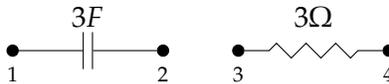
$$F(N) := \{(N, A, s, t, \ell) \mid \text{where } s, t: A \rightarrow N, \ell: E \rightarrow C\}.$$

On morphisms, F sends a function $f: N \rightarrow N'$ to the function

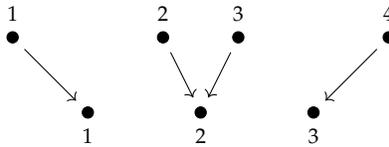
$$\begin{aligned} F(f): F(N) &\longrightarrow F(N'); \\ (N, A, s, t, \ell) &\longmapsto (N', A, s.f, t.f, \ell). \end{aligned}$$

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

Exercise 6.58. To understand this functor better, let $c \in F(\underline{4})$ be the circuit



and let $f: \underline{4} \rightarrow \underline{3}$ be the function



What is the circuit $F(f)(c)$? \diamond

We're trying to get a decoration functor (F, φ) and so far we have F . For the coherence maps $\varphi_{M,N}$ for finite sets M, N , we define

$$\begin{aligned} \varphi_{M,N}: F(M) \times F(N) &\longrightarrow F(M + N); \\ ((M, A, s, t, \ell), (N, A', s', t', \ell')) &\longmapsto (M + N, A + A', s + s', t + t', [\ell, \ell']). \end{aligned}$$

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

To understand concretely what this implies for composition of decorated cospans, we refer to Eq. (6.10) once more. First, the underlying cospan of the composite decorated cospan is just the composite of their underlying cospans. So, in Eq. (6.10), the composite decorated cospan decorates the cospan with apex $N +_B M$, a five element set. If we call the circuit in Eq. (6.8) c , and call the other circuit c' , then the composite decoration is $F[\iota_M, \iota_N](\varphi_{M,N}(c, c'))$.

Oh no, there's that horrendous formula again!⁵ But all $\varphi_{M,N}(c, c')$ says is to the

⁵We saw this formula on page 178.

disjoint union of circuit diagrams c and c' ; the result is a circuit on the seven element set $M + N$; let's call it c'' . Then $F([\iota_M, \iota_N])(c'')$ just transfers c'' along the canonical function $M + N \rightarrow M +_B N$. This might sound hard, but it's just like what you did in Exercise 6.58 when you computed $F(f)(c)$. Go look at it; you'll feel better.

6.5 Operads and their algebras

In Theorem 6.55 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 posets 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 your needs. For example in [VSL15] the authors wanted to compose 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”.

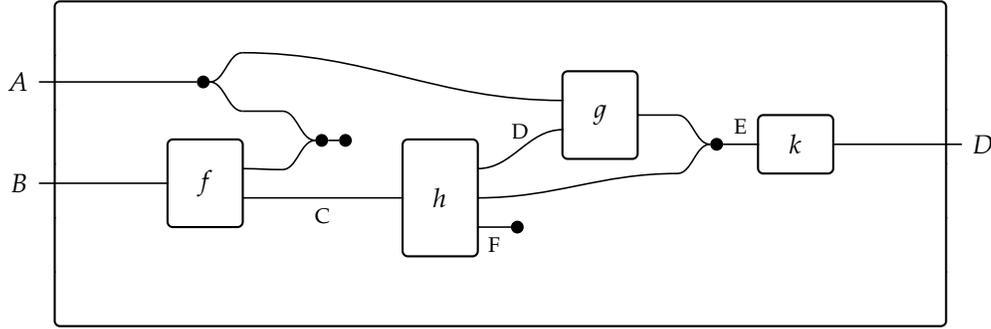
So to close our discussion of compositional structures, we want to quickly sketch something we can use as a sort of meta-compositional structure, known as an operad. We saw in Section 6.4.3 that we can build electric circuits from a symmetric monoidal functor $\mathbf{FinSet} \rightarrow \mathbf{Set}$.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

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 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 that of the parts, together with how they interact. 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, not ‘domain’ and ‘codomain’ boundaries. This is not to say the 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 models 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.

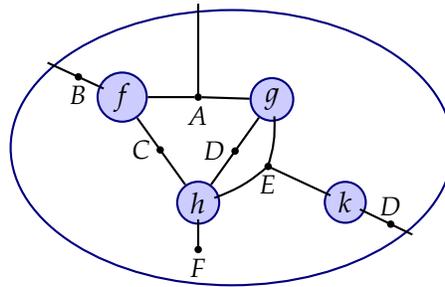
Let’s return to wiring diagrams for a second. We saw that wiring diagrams for hypergraph categories basically look like this:



(6.11)

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:



(6.12)

Do you see how diagrams Eq. (6.11) and Eq. (6.12) are actually exactly the same in terms of interconnection pattern? The only difference is that the latter does not have left/right distinction, which is what we said we wanted.

The downside is that the “boxes” f, g, h, k in Eq. (6.12) 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. (6.11)—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: they’re no longer morphisms in a category so 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 1.5.5. 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 $\mathcal{M} \rightarrow \mathbf{Set}$; see Remark 5.61. Similarly, we can view the category $\mathbf{Cospan}_{\mathbf{FinSet}}$ as some sort of

‘theory of hypergraph categories’, and so define hypergraph categories as functors $\mathbf{Cospan}_{\mathbf{FinSet}} \rightarrow \mathbf{Set}$.

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

Rough Definition 6.59. To specify an operad \mathcal{O} ,

- (i) one specifies a set T , whose elements are called *types*;
- (ii) for each tuple (t_1, \dots, t_n, t) of types, one specifies a set $\mathcal{O}(t_1, \dots, t_n; t)$, whose elements are called *operations of arity* $(t_1, \dots, t_n; t)$;
- (iii) for each pair of tuples (s_1, \dots, s_m, t_i) and (t_1, \dots, t_n, t) , one specifies a function

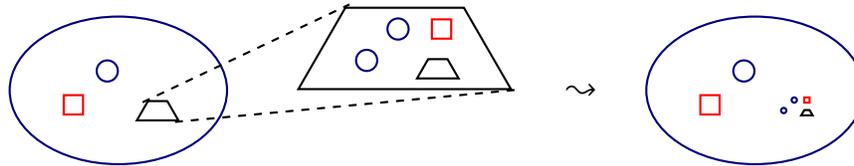
$$\circ_i: \mathcal{O}(s_1, \dots, s_m; t_i) \times \mathcal{O}(t_1, \dots, t_n; t) \rightarrow \mathcal{O}(t_1, \dots, t_{i-1}, s_1, \dots, s_m, t_{i+1}, \dots, t_n; t);$$

called *substitution*; and

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

These must obey generalized identity and associativity laws.⁶

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, operations that accept n arguments. If we take an operation f of arity m , and plug the output into the i th 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 $n - 1$ arguments to fill in g . Which operation of arity $m + n - 1$ do we get? This is described by the 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 this intuitive picture.



The types then allow us to specify the, well, types of the arguments—inputs—that each function takes. So making tea is an 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.

In defining an operad above we have used what is often called ‘circle i ’ notation. This is because we have asked for a composition rule that specifies the operation that results if you plug in an operation into the i th argument of a given operation. Another way of defining an operad is to specify functions that describe plugging in an operation into every argument in a given operation. There are a variety of different flavors of operad; we’ve given a flavor of the non-symmetric typed flavor.

⁶Often 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].

Example 6.60. 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, fights.

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]. \blacklozenge

6.5.2 Operads from symmetric monoidal categories

We will see in Definition 6.63 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 6.61. The operad **Set** of sets has

- (i) Sets X as types.
- (ii) Functions $X_1 \times \cdots \times X_n \rightarrow Y$ as operations of arity $(X_1, \dots, X_n; Y)$.
- (iii) Substitution defined by

$$\begin{aligned} (f \circ_i g)(x_1, \dots, x_{i-1}, w_1, \dots, w_m, x_{i+1}, \dots, x_n) \\ = g(x_1, \dots, x_{i-1}, f(w_1, \dots, w_m), x_{i+1}, \dots, x_n) \end{aligned}$$

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

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

- (iv) Identities $\text{id}_X \in \mathbf{Set}(X; X)$ are given by the identity function $\text{id}_X: X \rightarrow X$. \blacklozenge

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

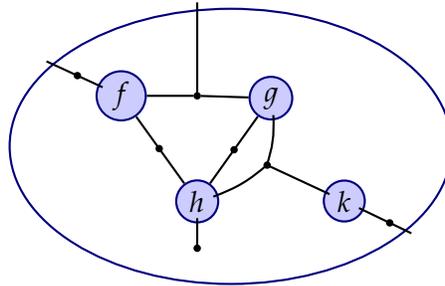
Example 6.62. The operad **Cospan** of finite-set cospans has

- (i) Natural numbers $a \in \mathbb{N}$ as types.
- (ii) Cospans $\underline{a}_1 + \cdots + \underline{a}_n \rightarrow \underline{p} \leftarrow \underline{b}$ of finite sets as operations of arity $(a_1, \dots, a_n; b)$.
- (iii) Substitution defined by pushout.

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

This is the operadic analogue of the category $\mathbf{Cospan}_{\mathbf{FinSet}}$.

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



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 it 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 \mathbf{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. \blacklozenge

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

Definition 6.63. For any symmetric monoidal category (C, I, \otimes) , there is an operad O_C , called the *operad underlying C* , defined as having:

- (i) $\text{Ob}(C)$ as types.
- (ii) morphisms $C_1 \otimes \cdots \otimes C_n \rightarrow D$ in C as the operations of arity $(C_1, \dots, C_n; D)$.
- (iii) substitution is defined by

$$(f \circ_i g) := f \circ (\text{id}, \dots, \text{id}, g, \text{id}, \dots, \text{id})$$

- (iv) identities $\text{id}_a \in O_C(a; a)$ 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 6.64. Suppose given two operads O and P with types T and U respectively. To specify an operad functor $F: O \rightarrow P$,

- (i) one specifies a function $f: T \rightarrow U$.
- (ii) For all arities $(t_1, \dots, t_n; t)$ in \mathcal{O} , one specifies a function

$$F: \mathcal{O}(t_1, \dots, t_n; t) \rightarrow \mathcal{P}(f(t_1), \dots, f(t_n); f(t))$$

such that composition and identities are preserved.

Just as set-valued functors $C \rightarrow \mathbf{Set}$ from any category C are of particular interest—and so have a special term, namely *presheaf on C* —so to are \mathbf{Set} -valued functors $\mathcal{O} \rightarrow \mathbf{Set}$ from any operad \mathcal{O} . Thus they too get a special name.

Definition 6.65. An *algebra* (or *model*) for an operad \mathcal{O} is an operad functor $F: \mathcal{O} \rightarrow \mathbf{Set}$.

We can think of functors $\mathcal{O} \rightarrow \mathbf{Set}$ as defining a set of possible ways to fill in 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 $F: \mathcal{O} \rightarrow \mathbf{Set}$ will take a type t and return a set $F(t)$. Moreover, given an operation (i.e., a wiring diagram) $f \in \mathcal{O}(t_1, \dots, t_n; t)$, 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 boundaries t_1, \dots, t_n respectively, and returns a circuit with boundary t .

This is reminiscent of the functor used to define decorated cospans, and indeed we get a similar result.

Proposition 6.66. *There is an equivalence between algebras for the operad **Cospan** and hypergraph props.*

In particular, the functor $(F, \varphi): (\mathbf{FinSet}, +) \rightarrow (\mathbf{Set}, \times)$ of Section 6.4.3 can be extended to a monoidal functor $(\mathbf{Cospan}_{\mathbf{FinSet}}, +) \rightarrow (\mathbf{Set}, \times)$ by a universal construction known as a Kan extension. These monoidal categories may be converted to operads in the manner described in Section 6.5.2, and the functor then converts into an operad algebra for **Cospan**. The resulting hypergraph prop agrees with the hypergraph prop constructed explicitly using decorated cospans.

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. [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 posets 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 described 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, then, one might refer to any of the introductory category theory textbooks we mentioned in Chapter 3.

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; Fon17; 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].

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

Logic of behavior: Sheaves, toposes, and internal languages

7.1 How can we prove our machine is safe?

Imagine you are trying to design a system of interacting components. You wouldn't be doing this if you didn't have a goal in mind: you want the system to do something, to behave in a certain way. In other words, you want to restrict its possibilities to a smaller set: you want the car to remain on the road, you want the temperature to remain in a range, you want the bridge to be safe for trucks. Out of all the possibilities, your system should only permit some.

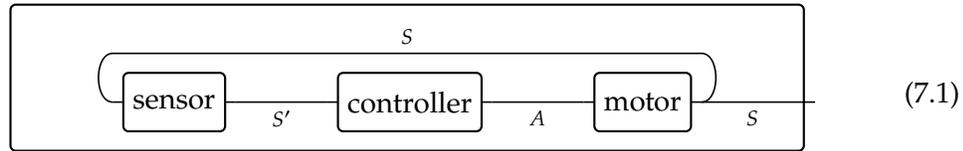
Since your system is made of components that interact in specified ways, the possible behavior of the whole—in any environment—is determined by the possible behaviors of each of its components in their local environments, together with the precise way in which they interact.¹ In this chapter, we will discuss a logic wherein one can describe general types of behavior that occur over time, and prove properties of a larger-scale system from the properties and interaction patterns of its components.

For example, suppose we want an autonomous vehicle to maintain a distance of at least some constant safe from another object. To do so, several components must interact: a sensor that approximates the real distance by an internal variable S' , a controller that uses S' to decide what action A to take, and a motor that moves the

¹ The well-known concept of emergence is not about possibilities, it is about prediction. Predicting the behavior of a system given predictions of its components is notoriously hard. The behavior of a double pendulum is chaotic—meaning extremely sensitive to initial conditions—whereas those of the two component pendulums are not. However, the set of possibilities for the double pendulum are completely understood: it is the set of possible angular positions and velocities of both arms. When we speak of a machine's properties in this chapter, we always mean the guarantees on its behaviors, not the probabilities involved, though the latter would certainly be an interesting thing to contemplate.

vehicle with an acceleration based on A . This in turn affects the real distance S , so there is a feedback loop.

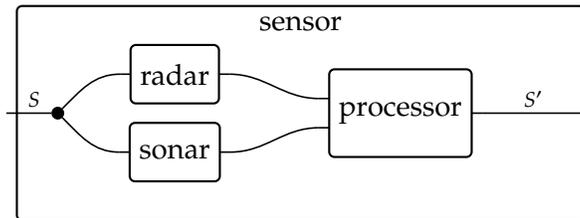
Consider the following model diagram:



In the diagram shown, the distance S is exposed by the exterior interface. This just means we imagine S as being a variable that other components of a larger system may want to interact with. We could have exposed no variables (making it a closed system) or we could have exposed A and/or S' as well.

In order for the system to ensure $S \geq \text{safe}$, we need each of the components to ensure a property of its own. But what are these components, “sensor, controller, motor”, and what do they do?

One way to think about any of the components is to open it up and see how it is put together; with a detailed study we may be able to say what it will do. For example, just as S was exposed in the diagram above, one could imagine opening up the sensor component box in Eq. (7.1) and seeing an interaction between subcomponents



This ability to zoom in and see a single unit as being composed of others is important for design. But at the end of the day, you eventually need to stop diving down and simply use the properties of the components in front of you to prove properties of the composed system. Your job is to design the system at a given level, taking the component properties of lower-level systems as given.

We will think of each component in terms of the relationship it maintains (through time) between the changing values on its ports. “Whenever I see a flash, I will increase pressure on the button”; this is a relationship I maintain between the changing values on my eye port and my finger port. We will make this more precise soon, but more examples should help. The sensor maintains a relationship between S and S' , e.g. that the real distance S and its internal representation S' differ by no more than 5cm. The controller maintains a relationship between S' and the action signal A , e.g. that if at any time $S < \text{safe}$, then within one second it will emit the signal $A = \text{go}$. The motor maintains a relationship between A and S , e.g. that A dictates the second derivative of S by the formula

$$(A = \text{go} \Rightarrow \ddot{S} > 1) \wedge (A = \text{stop} \Rightarrow \ddot{S} = 0). \quad (7.2)$$

If we want to prove properties of the whole interacting system, then—like Eq. (7.2)—the relationships maintained by each component need to be written in a formal logical language. From that basis, we can use standard proof techniques to combine properties of subsystems into properties of the whole. This is our objective in the present chapter.

We have said how component systems, wired together in some arrangement, create larger-scale systems. We have also said that, given the wiring arrangement, the behavioral properties of the component system behaviors dictate the behavioral properties of the whole. But what exactly are behavioral properties?

In this chapter, we want to give a formal language and semantics for a very general notion of behavior. Mathematics is itself a formal language; the usual style of mathematical modeling is to use any piece of this vast language at any time and for any reason. One uses “human understanding” to ensure that the different linguistic pieces are fitting together in an appropriate way when different systems are combined. For example, someone might say “replace the directed graph by its underlying symmetric simple graph, and use the second-smallest eigenvalue of its Laplacian to approximate the sparsest cut of the original graph.” Where the present work differs is that we want to find a domain-specific language for modeling behavior, any sort of behavior, and nothing but behavior. Unlike in the wide world of math, we want a setting where the only things that can be discussed are behaviors.

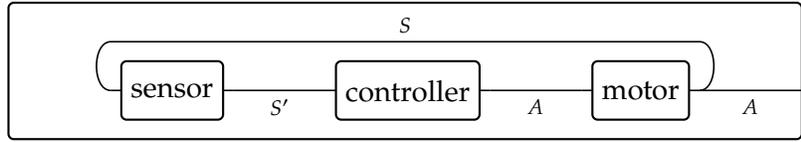
For this, we will construct what is called a *topos*, which is a special kind of category. Our topos, let’s call it **BT**, will have behavior types as its objects. An amazing fact about toposes² is that they come with an *internal language* that looks very much like the usual formal language of mathematics itself. Thus one can define graphs, groups, topological spaces, etc. in any topos. But in **BT**, what we call graphs will actually be graphs that change through time, and similarly what we call groups and spaces will actually be groups and spaces that change through time. Rather than living in the world of sets—where elements are constant through time—we will live in the world of behavior types, which are like “variable sets” that can change through time.

The topos **BT** not only has an internal language, but it has a mathematical semantics using the notion of sheaves. Technically, a sheaf is a certain sort of functor, but one can imagine it as a space of possibilities, like a space of possible behaviors. Every property we prove in our logic of behavior types will have meaning in this category of sheaves.

When discussing systems and components—such as sensors, controllers, motors, etc.—we mentioned behavior types; these will be the objects in the topos **BT**. Every wire in the picture below will stand for a behavior type, and every box X will stand for a behavioral property, a relation that X maintains between the changing values on its

²The plural of topos is often written *topoi*, rather than toposes. This seems a bit fancy for our taste. As Johnstone suggests in [Joh77], we might ask those who “persist in talking about topoi whether, when they go out for a ramble on a cold day, they carry supplies of hot tea with them in thermoi.” It’s all in good fun; either term is perfectly reasonable and well-accepted.

ports.



For example we could imagine that

- S (wire): The behavior of S over a time-interval $[a, b]$ is that of all continuous real-valued functions $[a, b] \rightarrow \mathbb{R}$.
- A (wire): The behavior of A over a time-interval $[a, b]$ is all piecewise constant functions, taking values in the finite set such as $\{\text{go}, \text{stop}\}$.
- controller (box): the relation $\{(S', A) \mid \text{Eq. (7.2)}\}$, i.e. all behavioral pairs (S', A) that conform to what our controller is supposed to do.

7.2 The category **Set** as an exemplar topos

We want to think about a very abstract sort of thing, called a topos, because we will see that behavior types form a topos. To get started, we begin with one of the easiest toposes to think about, namely the topos **Set** of sets. In this section we will discuss commonalities between sets and every other topos. We will go into some details about the category of sets, so as to give intuition for other toposes. In particular, we'll pay careful attention to the logic of sets, because we eventually want to understand the logic of behaviors.

Indeed, logic and sets are closely related. For example, the logical statement—more formally known as a predicate—`likes cats` defines a function from the set P of people to the set $\mathbb{B} = \{\text{false}, \text{true}\}$ of truth values, where $\text{Brendan} \in P$ maps to `true` because he likes cats whereas $\text{Ursula} \in P$ maps to `false` because she does not. Alternatively, `likes cats` also defines a subset of P , consisting of exactly the people that do like cats

$$\{p \in P \mid \text{likes cats}(p)\}.$$

In terms of these subsets, logical operations correspond to set operations, e.g. `AND` corresponds to intersection: indeed, the set of people mapped to `true` by `likes cats AND likes dogs` is equal to the intersection of the set `likes cats` and the set `likes dogs`.

We saw in Chapter 3 that such operations, which are examples of database queries, can be described in terms of limits and colimits in **Set**. Indeed, the category **Set** has many such structures and properties, which together make logic possible in that setting. In this section we want to identify these properties, and show how logical operations can be defined using them.

Why would we want to abstractly find such structures and properties? In the next section, we'll start our search for other categories that also have them. Such categories, called toposes, will be **Set**-like enough to do logic, but have much more complex and

interesting semantics. Indeed, we will discuss one whose logic allows us to reason not about properties of sets, but of about behavioral properties of very general machines.

7.2.1 Set-like properties enjoyed by any topos

Although we will not prove it in this book, toposes are categories that are similar to **Set** in many ways. Here are some facts that are true of any topos \mathcal{E} :

1. \mathcal{E} has all limits,
2. \mathcal{E} has all colimits,
3. \mathcal{E} is cartesian closed,
4. \mathcal{E} has epi-mono factorizations,
5. \mathcal{E} has a subobject classifier $1 \xrightarrow{\text{true}} \Omega$.

In particular, since **Set** is a topos, all of the above facts are true for $\mathcal{E} = \mathbf{Set}$. Our first goal is to briefly review these concepts, focusing most on the subobject classifier.

Limits and colimits We discussed limits and colimits briefly in Section 3.4.2, but the basic idea is that one can make new objects from old by taking products, using equations to define subobjects, forming disjoint unions, and taking quotients. There is also a terminal object 1 and an initial object 0 . One of the most important types of limits (resp. colimits) are pullbacks (resp. pushouts); see Example 3.83 and Eq. (6.3).

Suppose that \mathcal{C} is a category and consider the diagrams below:

$$\begin{array}{ccccc} A & \longrightarrow & B & \longrightarrow & C \\ \downarrow & & \downarrow & \lrcorner & \downarrow \\ D & \longrightarrow & E & \longrightarrow & F \end{array} \qquad \begin{array}{ccccc} A & \longrightarrow & B & \longrightarrow & C \\ \downarrow & \lrcorner & \downarrow & & \downarrow \\ D & \longrightarrow & E & \longrightarrow & F \end{array}$$

In the left-hand square, the corner symbol \lrcorner unambiguously means that the square (B, C, E, F) is a pullback. But in the right-hand square, does the corner symbol mean that (A, B, D, E) is a pullback or that (A, D, C, F) is a pullback? It's ambiguous, but as we next show, it becomes unambiguous if the right-hand square is a pullback.

Proposition 7.1. *In the commutative diagram below, suppose the (B, C, E, F) square is a pullback:*

$$\begin{array}{ccccc} A & \longrightarrow & B & \longrightarrow & C \\ \downarrow & \lrcorner & \downarrow & \lrcorner & \downarrow \\ D & \longrightarrow & E & \longrightarrow & F \end{array}$$

Then the (A, B, D, E) square is a pullback iff the (A, C, D, F) rectangle is a pullback.

Exercise 7.2. Prove Proposition 7.1 using the definition of limit from Section 3.4.2. \diamond

Epi-mono factorizations The abbreviation “epi” stands for *epimorphism*, and the abbreviation “mono” stands for *monomorphism*. Epimorphisms are maps that act like

surjections, and monomorphisms are maps that act like injections.³ We can define them formally in terms of pushouts and pullbacks.

Definition 7.3. Let C be a category, and let $f: A \rightarrow B$ be a morphism. It is called a *monomorphism* (resp. *epimorphism*) if the square to the left is a pullback (resp. the square to the right is a pushout):

$$\begin{array}{ccc} A & \xrightarrow{\text{id}_A} & A \\ \text{id}_A \downarrow & \lrcorner & \downarrow f \\ A & \xrightarrow{f} & B \end{array} \qquad \begin{array}{ccc} A & \xrightarrow{f} & B \\ f \downarrow & \lrcorner & \downarrow \text{id}_B \\ B & \xrightarrow{\text{id}_B} & B \end{array}$$

Exercise 7.4. Show that in **Set**, monomorphisms are just injections:

1. Show that if $f: A \rightarrow B$ is injective then it is a monomorphism.
2. Show that if f is a monomorphism then it is injective. ◇

Exercise 7.5. Let C be a category and suppose the following diagram is a pullback in C :

$$\begin{array}{ccc} A' & \longrightarrow & A \\ f' \downarrow & \lrcorner & \downarrow f \\ B' & \longrightarrow & B \end{array}$$

Use Proposition 7.1 to show that if f is a monomorphism, then so is f' . ◇

Now that we have defined epimorphisms and monomorphisms, we can say what epi-mono factorizations are. We say that a morphism $f: C \rightarrow D$ in \mathcal{E} has an epi-mono factorization if it has an “image”; that is, there is an object $\text{im}(f)$, an epimorphism $C \twoheadrightarrow \text{im}(f)$, and a monomorphism $\text{im}(f) \hookrightarrow D$, whose composite is f .

In **Set**, epimorphisms are surjections and monomorphisms are injections. Every function $f: C \rightarrow D$ may be factored as a surjective function onto its image $\text{im}(f) = \{f(c) \mid c \in C\}$, followed by the inclusion of this image into the codomain D . Moreover, this factorization is unique up to isomorphism.

This is the case in any topos \mathcal{E} : for any morphism $f: c \rightarrow d$, there is epimorphism e and a monomorphism m such that $f = e.m$ is their composite.

Cartesian closed A category C being cartesian closed means that C has a symmetric monoidal structure given by products, and it is monoidal closed with respect to this. (We previously saw monoidal closure in Definition 2.55 (for posets) and Proposition 4.42, as a corollary of compact closure.) Slightly more down-to-earth, cartesian closure means that for any two objects $c, d \in C$, there is a “hom-object” $d^c \in \mathcal{E}$ and a natural isomorphism

$$C(a \times c, d) \cong C(a, d^c) \tag{7.3}$$

³ Surjections are sometimes called “onto” and injections are sometimes called “one-to-one”, hence the Greek prefixes *epi* and *mono*.

Think of it this way: to use a function $f: (a \times c) \rightarrow d$, you must position two knobs, an a -knob and a c -knob, to get an answer in d . But if you fix the position of one knob now, say a , then you get a function that waits for you to position the other knob c to get an answer in d . This is the content of the isomorphism (7.3). We first saw this idea, called currying, in Example 3.58.

Subobject classifier The concept of a subobject classifier requires more attention, because its existence has huge consequences for a category \mathcal{C} . In particular, it creates the setting for a rich system of *higher order logic* to exist inside \mathcal{C} ; it does so by telling us the “truth values” for the topos. The higher order logic manifests in its fully glory when \mathcal{C} has finite limits and is cartesian closed, because these facts give rise to the logical operations on truth values.⁴ In particular, the higher order logic exists in any topos.

We will explain subobject classifiers in as much detail as we can; in fact, it will be our subject for the rest of Section 7.2.

7.2.2 The subobject classifier

Before giving the definition of subobject classifiers, recall that monomorphisms in **Set** are injections, and any injection $X \hookrightarrow Y$ is isomorphic to a subset of Y . This gives a simple and useful way to conceptualize monomorphisms into Y when reading the following definition: it will do no harm to think of them as subobjects of Y .

Definition 7.6. Let \mathcal{E} be a category with finite limits, i.e. with pullbacks and a terminal object 1 . A *subobject classifier* in \mathcal{E} is an object $\Omega \in \mathcal{E}$, together with a monomorphism $\text{true}: 1 \rightarrow \Omega$ satisfying the following property: for any objects X and Y and monomorphism $m: X \hookrightarrow Y$ in \mathcal{E} , there is a unique morphism $\ulcorner m \urcorner: Y \rightarrow \Omega$ such that the diagram on the left of Eq. (7.4) is a pullback in \mathcal{E} :

$$\begin{array}{ccc}
 X & \xrightarrow{!} & 1 \\
 \ulcorner m \urcorner \downarrow & \lrcorner & \downarrow \text{true} \\
 Y & \xrightarrow{\ulcorner m \urcorner} & \Omega
 \end{array}
 \qquad
 \begin{array}{ccc}
 \{Y \mid p\} & \xrightarrow{!} & 1 \\
 \ulcorner p \urcorner \downarrow & \lrcorner & \downarrow \text{true} \\
 Y & \xrightarrow{p} & \Omega
 \end{array}
 \tag{7.4}$$

We refer to $\ulcorner m \urcorner$ as the *characteristic map* of m , or we say that $\ulcorner m \urcorner$ *classifies* m . Conversely, given any map $p: Y \rightarrow \Omega$, we denote the pullback of true as on the right of Eq. (7.4).

A *predicate* on Y is a morphism $Y \rightarrow \Omega$.

Definition 7.6 is a bit difficult to get ones mind around, partly because it is hard to imagine its consequences. It is like a superdense nugget from outer space, and

⁴A category that has finite limits, is cartesian closed, and has a subobject classifier is called an *elementary topos*. We will not discuss these further, but they are the most general notion of topos in ordinary category theory. When someone says topos, you might ask “Grothendieck topos or elementary topos?”, because there does not seem to be widespread agreement on which is the default.

through scientific explorations in the latter half of the 20th century, we have found that it brings super powers to whichever categories possess it. We will explain some of the consequences below, but very quickly, the idea is the following.

The subobject classifier translates subobjects of Y , having arbitrary domain X , into maps from Y to a single fixed object Ω . We can replace our fantasy of the superdense nugget with a slightly more refined story: “any object Y understands itself—its parts and the logic of how they fit together—by consulting the oracle Ω and seeing what’s true.” Or to fully be precise but dry, “subobjects of Y are classified by predicates on Y .”

Let’s move from stories and slogans to concrete facts.

The subobject classifier in \mathbf{Set} Since \mathbf{Set} is a topos, it has a subobject classifier. It will be a set with supposedly wonderful properties; what set is it?

The subobject classifier in \mathbf{Set} is the set of booleans,

$$\Omega_{\mathbf{Set}} := \mathbb{B} = \{\text{true}, \text{false}\}. \quad (7.5)$$

So in \mathbf{Set} , the truth values are true and false.

By definition (Def. 7.6), the subobject classifier comes equipped with a morphism, generically called $\text{true}: 1 \rightarrow \Omega$; in the case of \mathbf{Set} it is played by the function $1 \rightarrow \{\text{true}, \text{false}\}$ that sends 1 to true. In other words, the morphism true is aptly named in this case.

For sets, monomorphism just means injection, as we mentioned above. So Definition 7.6 says that for any injective function $m: X \hookrightarrow Y$ between sets, we are supposed to be able to find a characteristic function $\ulcorner m \urcorner: Y \rightarrow \{\text{true}, \text{false}\}$ with some sort of pullback property. We propose the following definition of $\ulcorner m \urcorner$:

$$\ulcorner m \urcorner(y) := \begin{cases} \text{true} & \text{if } m(x) = y \text{ for some } x \in X \\ \text{false} & \text{otherwise} \end{cases}$$

In other words, if we think of X as a subobject of Y , then $\ulcorner m \urcorner(y)$ is true iff $y \in X$.

In particular, the subobject classifier property turns subsets $X \subseteq Y$ into functions $p: Y \rightarrow \mathbb{B}$, and vice versa. How it works is encoded in Definition 7.6, but the basic idea is that X will be the set of all things in Y that p sends to true:

$$X = \{y \in Y \mid p(y) = \text{true}\}. \quad (7.6)$$

This might help explain our abstract notation $\{Y \mid p\}$ in Eq. (7.4).

Exercise 7.7. Let $X = \mathbb{N} = \{0, 1, 2, \dots\}$ and $Y = \mathbb{Z} = \{\dots, -1, 0, 1, 2, \dots\}$; we have $X \subseteq Y$, so consider it as a monomorphism $m: X \rightarrow Y$. It has a characteristic function $\ulcorner m \urcorner: Y \rightarrow \mathbb{B}$, as in Definition 7.6.

1. What is $\ulcorner m \urcorner(-5) \in \mathbb{B}$?
2. What is $\ulcorner m \urcorner(0) \in \mathbb{B}$?

◇

- Exercise 7.8.* 1. Consider the identity function $\text{id}_{\mathbb{N}}: \mathbb{N} \rightarrow \mathbb{N}$. It is an injection, so it has a characteristic function $\ulcorner \text{id}_{\mathbb{N}} \urcorner: \mathbb{N} \rightarrow \mathbb{B}$. Give a concrete description of $\ulcorner \text{id}_{\mathbb{N}} \urcorner$, i.e. its exact value for each natural number $n \in \mathbb{N}$.
2. Consider the unique function $!_{\mathbb{N}}: \emptyset \rightarrow \mathbb{N}$ from the empty set. Give a concrete description of $\ulcorner !_{\mathbb{N}} \urcorner$. \diamond

7.2.3 Logic in the topos **Set**

As we said above, the subobject classifier of any topos \mathcal{E} gives the setting in which to do logic. Before we explain a bit about how topos logic works in general, we continue to work concretely by focusing on the topos **Set**.

Obtaining the AND operation Consider the function $1 \rightarrow \mathbb{B} \times \mathbb{B}$ picking out the element $(\text{true}, \text{true})$. This is a monomorphism, so it defines a characteristic function $\ulcorner (\text{true}, \text{true}) \urcorner: \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$. What function is it? By Eq. (7.6) the only element of $\mathbb{B} \times \mathbb{B}$ that can be sent to true is $(\text{true}, \text{true})$. Thus $\ulcorner (\text{true}, \text{true}) \urcorner(P, Q) \in \mathbb{B}$ must be given by the following truth table

P	Q	$\ulcorner (\text{true}, \text{true}) \urcorner(P, Q)$
true	true	true
true	false	false
false	true	false
false	false	false

This is the truth table for the AND of P and Q , i.e. for $P \wedge Q$. In other words, $\ulcorner (\text{true}, \text{true}) \urcorner = \wedge$. Note that this defines \wedge as a function $\wedge: \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$, and we define the usual infix notation $x \wedge y := \wedge(x, y)$.

Obtaining the OR operation Let's go backwards this time. The truth table for the OR of P and Q , i.e. that of the function $\vee: \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$ defining OR, is:

P	Q	$P \vee Q$
true	true	true
true	false	true
false	true	true
false	false	false

(7.7)

If we wanted to obtain this function as the characteristic function $\ulcorner m \urcorner$ of some subset $m: X \subseteq \mathbb{B} \times \mathbb{B}$, what subset would X be? By Eq. (7.6), X should be the set of $y \in Y$ that are sent to true . Thus m is the characteristic map for the three element subset

$$X = \{(\text{true}, \text{true}), (\text{true}, \text{false}), (\text{false}, \text{true})\} \subseteq \mathbb{B} \times \mathbb{B}.$$

To generalize this, we want a way of thinking of X only in terms of properties listed at the beginning of Section 7.2.1. In fact, one can think of X as the union of $\{\text{true}\} \times \mathbb{B}$ and

$\mathbb{B} \times \{\mathbf{true}\}$ —a colimit of limits involving the subobject classifier and terminal object. This constructs an analogous object to X in any topos.

Exercise 7.9. Every boolean has a negation, $\neg\mathbf{false} = \mathbf{true}$ and $\neg\mathbf{true} = \mathbf{false}$. The function $\neg: \mathbf{Bool} \rightarrow \mathbf{Bool}$ is the characteristic function of some thing, (*?*).

1. What sort of thing should (*?*) be? For example, should \neg be the characteristic function of an object? A topos? A morphism? A subobject? A pullback diagram?
2. Now that you know the sort of thing (*?*) is, which thing of that sort is it? \diamond

Exercise 7.10. Given two booleans P, Q , define $Q \Rightarrow P$ to mean $Q = (P \wedge Q)$.

1. Write down the truth table for the statement $Q = (P \wedge Q)$:

P	Q	$P \wedge Q$	$Q = (P \wedge Q)$
true	true	?	?
true	false	?	?
false	true	?	?
false	false	?	?

2. If you already have an idea what $Q \Rightarrow P$ should mean, does it agree with the last column of table above?
3. What is the characteristic function $m: \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$ for $Q \Rightarrow P$?
4. What subobject does m classify? \diamond

Exercise 7.11. Consider the sets $E := \{n \in \mathbb{N} \mid n \text{ is even}\}$, $P := \{n \in \mathbb{N} \mid n \text{ is prime}\}$, and $T := \{n \in \mathbb{N} \mid n \geq 10\}$. Each is a subset of \mathbb{N} , so defines a function $\mathbb{N} \rightarrow \mathbb{B}$.

1. What is $\ulcorner E \urcorner(17)$?
2. What is $\ulcorner P \urcorner(17)$?
3. What is $\ulcorner T \urcorner(17)$?
4. Name the smallest three elements in the set classified by $(\ulcorner T \urcorner \wedge \ulcorner P \urcorner) \vee \ulcorner E \urcorner$. \diamond

Review Let’s take stock of where we are and where we’re going. In Section 7.1, we set out our goal of proving properties about behavior, and we said that topos theory is a good mathematical setting for that. We are now at the end of Section 7.2, which was about **Set** as an exemplar topos. What happened?

In Section 7.2.1, we talked about properties of **Set** that are enjoyed by any topos: limits and colimits, cartesian closure, epi-mono factorizations, and subobject classifiers. Then in Section 7.2.2 we launched into thinking about the subobject classifier in general and in the specific topos **Set**, where it is the set \mathbb{B} of booleans because any subset of Y is classified by a specific predicate $p: Y \rightarrow \mathbb{B}$. Finally, in Section 7.2.3 we discussed how to understand logic in terms of Ω : there are various maps $\Omega \times \Omega \rightarrow \Omega$ or $\Omega \rightarrow \Omega$ etc., which serve as logical connectives such as $\wedge, \vee, \Rightarrow, \neg$, etc. These are operations on truth values.

We have talked a lot about toposes, but we’ve only seen one so far: the category of sets. But we’ve actually seen more without knowing it: the category **C-Inst** of instances on any database schema from Definition 3.50 is a topos. Such toposes are

called *presheaf toposes* and are interesting, but we will focus on *sheaf toposes*, because our topos of behavior types will be a sheaf topos.

Sheaves are very interesting, but highly abstract mathematical objects. They are not for the faint of mathematical heart (those who are faint of physical heart are welcome to proceed).

7.3 Sheaves

Sheaf theory began before category theory, e.g. in the form of something called “local coefficient systems for homology groups”. However its modern formulation in terms of functors and sites is due to Grothendieck, who also invented toposes.

The basic idea is that rather than study spaces, we should study what happens *on* spaces. A space is merely the “site” at which things happen. For example, if we think of the plane \mathbb{R}^2 as a space, we might examine only points and regions. But if we think of \mathbb{R}^2 as a site where things happen, then we might think of things like weather systems throughout the plane, or sand dunes, or trajectories and flows of material. There are many sorts of things that can happen on a space, and these are the sheaves: a sheaf is roughly “a sort of thing that can happen on the space”. If we want to think about points or regions from the sheaf perspective, we think about how these regions of space give us different points of view on what’s happening. That is, it’s all about what happens on a space: the parts of the space are just perspectives from which to watch.

This is reminiscent of databases. The schema of a database is not the interesting part; the data is what’s interesting. Indeed, the schema of a database is a site and the category of all instances on it is a topos. In general, we can think of any small category C as a site; the corresponding topos is the category of functors $C^{\text{op}} \rightarrow \mathbf{Set}$.⁵

Did you notice that we just introduced a huge class of toposes? For any category C , we said there is a topos of presheaves on it. So before we go on to sheaves, let’s discuss this preliminary topic of presheaves. We will begin to develop some terminology and ways of thinking that will generalize to sheaves.

7.3.1 Presheaves

Recall the definition of functor and natural transformation from Section 3.3. Presheaves are just functors, but they have special terminology that leads us to think about them in a certain geometric way.

Definition 7.12. Let C be a small category. A *presheaf* P on C is a functor $P: C^{\text{op}} \rightarrow \mathbf{Set}$. To each object $c \in C$, we refer to the set $P(c)$ as *the set of sections of P over c* . To each morphism $f: c' \rightarrow c$, we refer to the function $P(f): P(c) \rightarrow P(c')$ as *the restriction map along f* . For any section $s \in P(c)$, we may denote $P(f)(s) \in P(c')$, i.e. its restriction along f , by $s|_f$.

⁵The category of functors $C \rightarrow \mathbf{Set}$ is also a topos: use C^{op} as the defining site.

If P and Q are presheaves, a *morphism* $f: P \rightarrow Q$ between them is a natural transformation of functors $C^{\text{op}} \rightarrow \mathbf{Set}$.

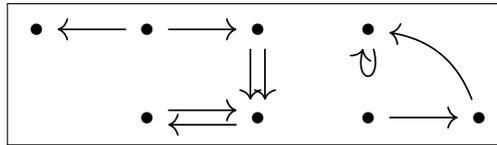
Example 7.13. Let **ArShp** be the category shown below:

$$\mathbf{ArShp} := \boxed{\begin{array}{ccc} \text{Vertex} & \begin{array}{c} \xrightarrow{\text{src}} \\ \xrightarrow{\text{tgt}} \end{array} & \text{Pure Arrow} \\ \bullet & & \bullet \end{array}}$$

The reason we call our category **ArShp** is that we can imagine of it as an “arrow shape”.

$$\begin{array}{c} \text{Vertex} := \boxed{\bullet} \\ \text{src} \quad \text{tgt} \\ \text{Pure Arrow} := \boxed{\longrightarrow} \end{array} \tag{7.8}$$

A presheaf on **ArShp** is a functor $I: \mathbf{ArShp}^{\text{op}} \rightarrow \mathbf{Set}$, which is a database instance on **ArShp**^{op}. Note that **ArShp**^{op} is what we called **Gr** in Section 3.3.5; there we showed that database instances on **Gr**—i.e. presheaves on **ArShp**— are just directed graphs, e.g.



Thinking of presheaves on any category C , it often makes sense to imagine the objects of C as shapes of some sort, and the morphisms of C as continuous maps between shapes, just like we did for the arrow shape in Eq. (7.8). In that context, one can think of a presheaf P as a kind of lego construction: P is built out of the shapes in C , connected together using the morphisms in C . In the case where C is the arrow shape, a presheaf is a graph. So this would say that a graph is a sort of lego construction, built out of vertices and arrows connected together using the inclusion of a vertex as the source or target of an arrow. Can you see it?

This statement can be made pretty precise; though we cannot go through it here, the above lego idea is summarized by the formal statement that “the category of presheaves on C is the free colimit completion of C .” \blacklozenge

However one thinks of presheaves—in terms of lego assemblies or database instances—they’re relatively straightforward. The difference between presheaves and sheaves is that sheaves take into account some sort of “covering information”. The trivial notion of covering is to say that every object covers itself and nothing more; if one uses this trivial covering, presheaves and sheaves are the same thing. In our context we will need a non-trivial notion of covering, so sheaves and presheaves will be slightly different. Our next goal is to understand sheaves on a topological space.

7.3.2 Topological spaces

We said in Section 7.3 that, rather than study spaces, we consider spaces as mere “sites” on which things happen. We also said the things that can happen on a space are called sheaves, and always form a type of category called a topos. To define a topos of sheaves, we must start with the site on which they exist.

Sites are very general mathematical objects, and we will not make them precise in this book. However, one of the easiest sorts of sites to think about are those coming from topological spaces: every topological space naturally has the structure of a site. We’ve talked about spaces for a while without making them precise; let’s do so now.

Definition 7.14. Let X be a set, and let $P(X) = \{U \subseteq X\}$ denote its set of subsets. A *topology* on X is a subset $\mathbf{Op} \subseteq P(X)$, elements of which we call *open sets*,⁶ satisfying the following conditions:

- (a) Whole set: the subset $X \subseteq X$ is open, i.e. $X \in \mathbf{Op}$.
- (b) Binary intersections: if $U, V \in \mathbf{Op}$ then $(U \cap V) \in \mathbf{Op}$.
- (c) Arbitrary unions: if I is a set and if we are given an open set $U_i \in \mathbf{Op}$ for each i , then their union is also open, $(\bigcup_{i \in I} U_i) \in \mathbf{Op}$. We interpret the particular case where $I = \emptyset$ to mean that the empty set is open: $\emptyset \in \mathbf{Op}$.

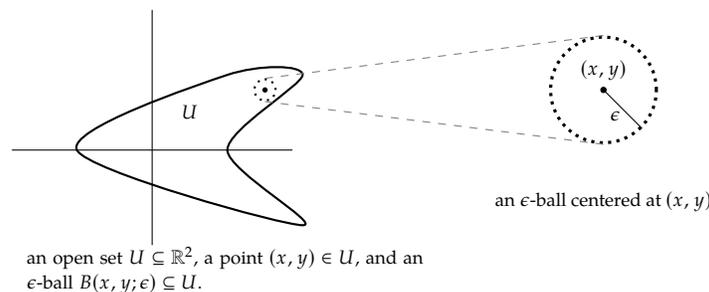
If $U = \bigcup_{i \in I} U_i$, we say that $(U_i)_{i \in I}$ covers U .

A pair (X, \mathbf{Op}) , where X is a set and \mathbf{Op} is a topology on X , is called a *topological space*.

A *continuous function* between topological spaces (X, \mathbf{Op}_X) and (Y, \mathbf{Op}_Y) is a function $f: X \rightarrow Y$ such that for every $U \in \mathbf{Op}_Y$, the preimage $f^{-1}(U)$ is in $\mathbf{Op}(X)$.

At the very end of Section 7.3.1 we mentioned how sheaves differ from presheaves in that they take into account “covering information”. The notion of covering an open set by a union of other open sets was defined in Definition 7.14, and it will come into play when we define sheaves in Definition 7.24.

Example 7.15. The usual topology \mathbf{Op} on \mathbb{R}^2 is based on “ ϵ -balls”. For any $\epsilon \in \mathbb{R}$ with $\epsilon > 0$, and any point $(x, y) \in \mathbb{R}^2$, let $B(x, y; \epsilon) = \{(x', y') \in \mathbb{R}^2 \mid (x - x')^2 + (y - y')^2 < \epsilon^2\}$: it is the set of all points within ϵ of (x, y) , and we call it the ϵ -ball centered at (x, y) .



⁶In other words, we refer to a subset $U \subseteq X$ as *open* if $U \in \mathbf{Op}$.

For an arbitrary subset $U \subseteq \mathbb{R}^2$, we call it open and put it in \mathbf{Op} if, for every $(x, y) \in U$ there exists an $\epsilon > 0$ such that $B(x, y; \epsilon) \subseteq U$.

The same idea works for any metric space X (Definition 2.32): it can be considered as a topological space where the open sets are subsets U such that for any $p \in U$ there is an ϵ -ball centered at p and contained in U . \blacklozenge

- Exercise 7.16.*
1. Define the usual topology on \mathbb{R} using ϵ -balls as above.
 2. Find three open sets U_1, U_2 , and U in \mathbb{R} , such that $(U_i)_{i \in \{1,2\}}$ covers U .
 3. Find an open set U and a collection $(U_i)_{i \in I}$ of opens sets where I is infinite, such that $(U_i)_{i \in I}$ covers U . \blacklozenge

Example 7.17. For any set X , there is “coarsest” topology, having as few open sets as possible: $\mathbf{Op}_{\text{crse}} = (\emptyset, X)$. There is also a “finest” topology, having as many open sets as possible: $\mathbf{Op}_{\text{fine}} = \mathbf{P}(X)$. The latter, $(X, \mathbf{P}(X))$ is called the *discrete space on the set X* . \blacklozenge

- Exercise 7.18.*
1. Verify that for any set X , what we called $\mathbf{Op}_{\text{crse}}$ in Example 7.17 really is a topology, i.e. satisfies the conditions of Definition 7.14.
 2. Verify also that $\mathbf{Op}_{\text{fine}}$ really is a topology.
 3. Show that if $(X, \mathbf{P}(X))$ is discrete and (Y, \mathbf{Op}_Y) is any topological space, then every function $X \rightarrow Y$ is continuous. \blacklozenge

Example 7.19. There are four topologies possible on $X = \{1, 2\}$. Two are $\mathbf{Op}_{\text{crse}}$ and $\mathbf{Op}_{\text{fine}}$ from Example 7.17. The other two are:

$$\mathbf{Op}_1 := \{\emptyset, \{1\}, X\} \quad \text{and} \quad \mathbf{Op}_2 := \{\emptyset, \{2\}, X\}$$

The two topological spaces $(\{1, 2\}, \mathbf{Op}_1)$ and $(\{1, 2\}, \mathbf{Op}_2)$ are isomorphic; either one can be called *the Sierpinski space*. \blacklozenge

The open sets of a topological space form a poset, \mathbf{Op} Given a topological space (X, \mathbf{Op}) , the set \mathbf{Op} has the structure of a poset using the subset relation, (\mathbf{Op}, \subseteq) . It is reflexive because $U \subseteq U$ for any $U \in \mathbf{Op}$, and it is transitive because if $U \subseteq V$ and $V \subseteq W$ then $U \subseteq W$.

Recall from Section 3.2.3 that we can regard any poset, and hence \mathbf{Op} , as a category: its objects are the open sets U and for any U, V the set of morphisms $\mathbf{Op}(U, V)$ is empty if $U \not\subseteq V$ and it has one element if $U \subseteq V$.

Exercise 7.20. Recall the Sierpinski space, say (X, \mathbf{Op}_1) from Example 7.19.

1. Write down the Hasse diagram for its poset of opens.
2. Write down all the coverings. \blacklozenge

Exercise 7.21. Given any topological space (X, \mathbf{Op}) , any subset $Y \subseteq X$ can be given the *subspace topology*, call it $\mathbf{Op}_{? \cap Y}$. This topology defines any $A \subseteq Y$ to be open, $A \in \mathbf{Op}_{? \cap Y}$, if there is an open set $B \in \mathbf{Op}$ such that $A = B \cap Y$.

1. Find a $B \in \mathbf{Op}$ that shows that the whole set Y is open, i.e. $Y \in \mathbf{Op}_{? \cap Y}$.

2. Show that $\mathbf{Op}_{\cap Y}$ is a topology in the sense of Definition 7.14.⁷
3. Show that the inclusion function $Y \hookrightarrow X$ is a continuous function. \diamond

Remark 7.22. Suppose (X, \mathbf{Op}) is a topological space, and consider the poset (\mathbf{Op}, \subseteq) of open sets. It turns out that $(\mathbf{Op}, \subseteq, X, \cap)$ is always a quantale in the sense of Definition 2.55, though we will not need that fact.

Exercise 7.23. In Sections 2.3.2 and 2.3.3 we discussed how **Bool**-categories are posets and **Cost**-categories are Lawvere metric spaces, and in Section 2.3.4 we imagined interpretations of \mathcal{V} -categories for other quantales \mathcal{V} .

If (X, \mathbf{Op}) is a topological space and \mathcal{V} the corresponding quantale as in Remark 7.22, how might we imagine a \mathcal{V} -category? \diamond

7.3.3 Sheaves on topological spaces

To summarize where we are, a topological space (X, \mathbf{Op}) is a set X together with a bunch of subsets we call “open”; these open subsets form a poset—and hence category—denoted \mathbf{Op} . Sheaves on X will be presheaves on \mathbf{Op} with a special property, aptly named the “sheaf condition”.

Recall the terminology and notation for presheaves: a presheaf on \mathbf{Op} is a functor $P: \mathbf{Op}^{\text{op}} \rightarrow \mathbf{Set}$. Thus to every $U \in \mathbf{Op}$ we have a set $P(U)$, called the set of *sections over U* , and to every $V \subseteq U$ we have a function $P(U) \rightarrow P(V)$ called the *restriction*. If $s \in P(U)$ is a section over U , we may denote its restriction to V by $s|_V$. Recall that we say a collection of open sets $(U_i)_{i \in I}$ *covers* an open set U if $U = \bigcup_{i \in I} U_i$.

We are now ready to give the full definition, which comes in several waves.

Definition 7.24. Let (X, \mathbf{Op}) be a topological space, and let $P: \mathbf{Op}^{\text{op}} \rightarrow \mathbf{Set}$ be a presheaf on \mathbf{Op} .

Let $(U_i)_{i \in I}$ be a collection of open sets $U_i \in \mathbf{Op}$ covering U . A *matching family* $(s_i)_{i \in I}$ of P -sections over $(U_i)_{i \in I}$ consists of a section $s_i \in P(U_i)$ for each $i \in I$, such that for every $i, j \in I$ we have

$$s_i|_{U_i \cap U_j} = s_j|_{U_i \cap U_j}.$$

Given a matching family $(s_i)_{i \in I}$ for the cover $U = \bigcup_{i \in I} U_i$, we say that $s \in P(U)$ is a *gluing*, or *glued section*, of the matching family if $s|_{U_i} = s_i$ holds for all $i \in I$.

If there exists a unique gluing $s \in P(U)$ for every matching family $(s_i)_{i \in I}$, we say that P *satisfies the sheaf condition for the cover* $U = \bigcup_{i \in I} U_i$. If P satisfies the sheaf condition for every cover, we say that P is a *sheaf* on (X, \mathbf{Op}) .

Thus a sheaf is just a presheaf satisfying the sheaf condition for every open cover. If P and Q are sheaves, then a *morphism* $f: P \rightarrow Q$ between these sheaves is just a morphism—that is, a natural transformation—between their underlying presheaves. We denote by $\mathbf{Shv}(X, \mathbf{Op})$ the category of sheaves on X .

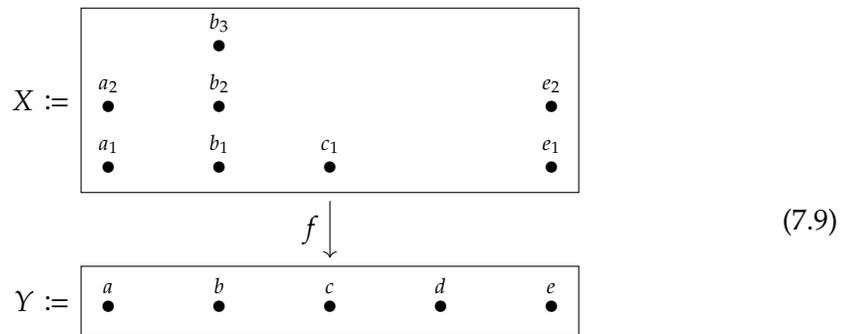
⁷Hint 1: for any set I , collection of sets $(U_i)_{i \in I}$ with $U_i \subseteq X$, and set $V \subseteq X$, one has $(\bigcup_{i \in I} U_i) \cap V = \bigcup_{i \in I} (U_i \cap V)$. Hint 2: for any $U, V, W \subseteq X$, one has $(U \cap W) \cap (V \cap W) = (U \cap V) \cap W$.

Example 7.25. Here is a funny special case to which the notion of matching family applies. We do not give this example for intuition, but because it’s an important and easy-to-miss case. Just like the sum of no numbers is 0 and the product of no numbers is 1, the union of no sets is the empty set. Thus if we take $U = \emptyset \subseteq X$ and $I = \emptyset$, then the empty collection of subsets (one for each $i \in I$, of which there are none) covers U . In this case the empty tuple $()$ counts a matching family of sections, and it is the only matching family for the empty cover.

In other words, in order for a presheaf $P: \mathbf{Op}^{\text{op}} \rightarrow \mathbf{Set}$ to be a sheaf, a necessary (but rarely sufficient) condition is that $P(\emptyset) \cong \{()\}$, i.e. $P(\emptyset)$ must be a set with one element. ♦

Extended example: sections of a function This example is for intuition, and gives a case where the “section” and “restriction” terminology are easy to visualize.

Consider the function $f: X \rightarrow Y$ shown below, where each element of X is mapped to the element of Y immediately below it. For example, $f(a_1) = f(a_2) = a$, $f(b_1) = b$, and so on.



For each point $y \in Y$, the set $f^{-1}(y) \subseteq X$ above it is called the *fiber over y* .

- Exercise 7.26.*
1. What is the fiber over a ?
 2. What is the fiber over c ?
 3. What is the fiber over d ?

♦

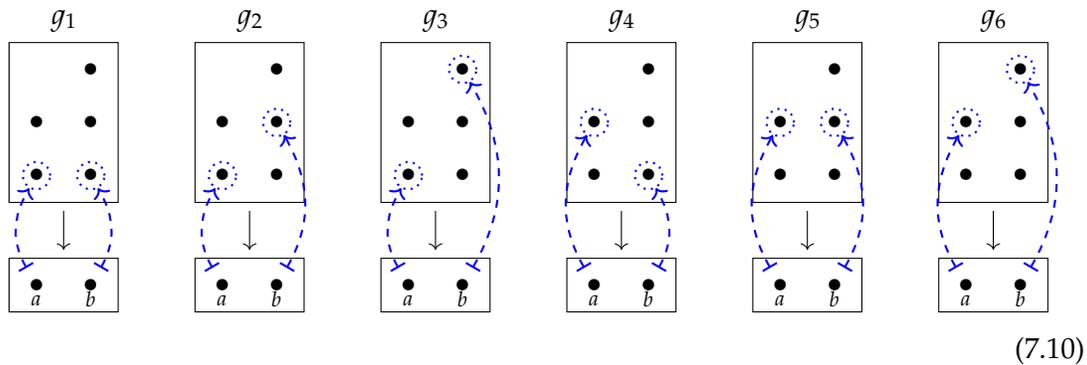
Let’s consider X and Y as discrete topological spaces, so every subset is open, and f is automatically continuous (see Exercise 7.18). We will use f to build a sheaf on Y . To do this, we begin by building a presheaf—i.e. a functor $\mathbf{Sec}_f: \mathbf{Op}(Y) \rightarrow \mathbf{Set}$ —and then we’ll prove it’s a sheaf.

Define the presheaf \mathbf{Sec}_f on an arbitrary subset $U \subseteq Y$ by:

$$\mathbf{Sec}_f(U) := \{g: U \rightarrow X \mid (g.f)(u) = u \text{ for all } u \in U\}.$$

One might describe it as the set of all ways to pick a “cross-section” of X over U . That is an element $g \in \mathbf{Sec}_f(U)$ chooses an element in the fiber over each $u \in U$.

As an example, let's say $U = \{a, b\}$. How many such g 's are there in $\text{Sec}_f(U)$? To answer this, let's clip the picture (7.9) and look only at the relevant part:

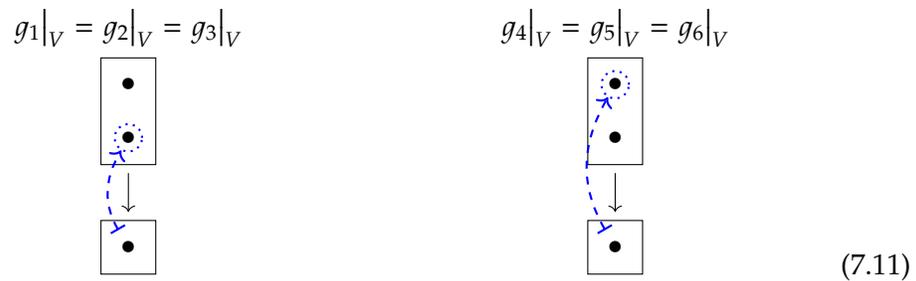


Looking at the picture (7.10), do you see how we get all cross-sections of f over U ?

Exercise 7.27. Refer to Eq. (7.9).

1. Let $V_1 = \{a, b, c\}$. Draw all the sections over it, i.e. all elements of $\text{Sec}_f(V_1)$, as we did in Eq. (7.10).
2. Let $V_2 = \{a, b, c, d\}$. Again draw all the sections, $\text{Sec}_f(V_2)$.
3. Let $V_3 = \{a, b, d, e\}$. How many sections (elements of $\text{Sec}_f(V_3)$) are there? \diamond

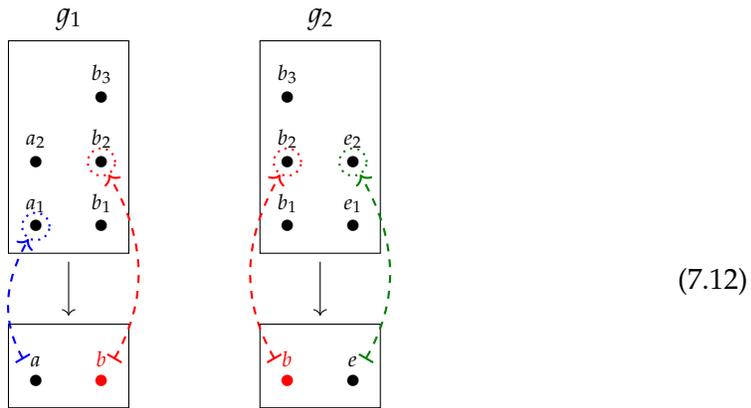
By now you should understand the sections of $\text{Sec}_f(U)$ for various $U \subseteq X$, but a presheaf also has a restriction maps for every subset $V \subseteq U$. Luckily, the restriction maps are easy: if $V \subseteq U$, say $V = \{a\}$ and $U = \{a, b\}$, then given a section g as in Eq. (7.10), we get a section over V by “restricting” our attention to what g does on $\{a\}$.



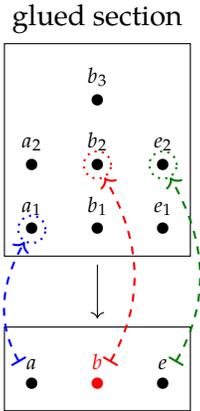
- Exercise 7.28.*
1. Write out the sets of sections $\text{Sec}_f(\{a, b, c\})$ and $\text{Sec}_f(\{a, c\})$.
 2. Draw lines from the first to the second to indicate the restriction map. \diamond

Now we have understood Sec_f as a presheaf; we next explain how to see that it is a sheaf, i.e. that it satisfies the sheaf condition for every cover. To understand the sheaf condition, consider the set $U_1 = \{a, b\}$ and $U_2 = \{b, e\}$. These cover the set $U = \{a, b, e\} = U_1 \cup U_2$. By Definition 7.24, a matching family for this cover consists of a section over U_1 and a section over U_2 that agree on the overlap set, $U_1 \cap U_2 = \{b\}$.

So consider $g_1 \in \text{Sec}_f(U_1)$ and $g_2 \in \text{Sec}_f(U_2)$ shown below.



Since sections g_1 and g_2 agree on the overlap—they both send b to b_2 —the two sections shown in Eq. (7.12) can be glued to form a single section over $U = \{a, b, e\}$:



Exercise 7.29. Again let $U_1 = \{a, b\}$ and $U_2 = \{b, c\}$, so the overlap is $U_1 \cap U_2 = \{b\}$.

1. Find a section $g_1 \in \text{Sec}_f(U_1)$ and a section $g_2 \in \text{Sec}_f(U_2)$ that *do not* agree on the overlap.
2. For your answer (g_1, g_2) in part 1, can you find a section $g \in \text{Sec}_f(U_1 \cup U_2)$ such that $g|_{U_1} = g_1$ and $g|_{U_2} = g_2$?
3. Find a section $h_1 \in \text{Sec}_f(U_1)$ and a section $h_2 \in \text{Sec}_f(U_2)$ that *do* agree on the overlap, but which are different than our choice in Eq. (7.12).
4. Can you find a section $h \in \text{Sec}_f(U_1 \cup U_2)$ such that $h|_{U_1} = h_1$ and $h|_{U_2} = h_2$? \diamond

Other examples of sheaves The extended example above generalizes to any continuous function between topological spaces.

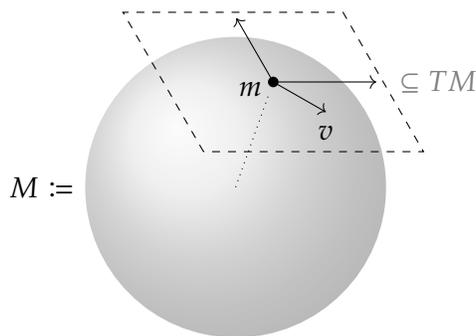
Example 7.30. Let $f: (X, \mathbf{Op}_X) \rightarrow (Y, \mathbf{Op}_Y)$ be a continuous function. Consider the functor $\text{Sec}_f: \mathbf{Op}_Y^{\text{op}} \rightarrow \mathbf{Set}$ given by

$$\text{Sec}_f(U) := \{g: U \rightarrow X \mid g \text{ is continuous and } (g.f)(u) = u \text{ for all } u \in U\},$$

The morphisms of \mathbf{Op}_Y are inclusions $V \subseteq U$. Given $g: U \rightarrow X$ and $V \subseteq U$, what we call the restriction of g to V is the usual thing we mean by restriction, the same as it was in Eq. (7.11). One can again check that \mathbf{Sec}_f is a sheaf. \blacklozenge

Example 7.31. A nice example of a sheaf on a space M is that of vector fields on M . If you calculate the wind velocity at every point on Earth, you will have what's called a vector field on Earth. If you know the wind velocity at every point in Afghanistan and I know the wind velocity at every point in Pakistan, and our calculations agree around the border, then we can glue our information together to get the wind velocity over the union of the two countries. All possible wind velocity fields over all possible open sets of the Earth's surface together form the sheaf of vector fields.

Let's say this a bit more formally. A manifold M —you can just imagine a sphere such as the Earth's surface—always has something called a tangent bundle. It is a space TM whose points are pairs (m, v) , where $m \in M$ is a point in the manifold and v is a tangent vector there. Here's a picture of one tangent plane on a sphere:



The tangent bundle TM includes the whole tangent plane shown above—including the three vectors drawn on it—as well as the tangent plane at every other point on the sphere.

The tangent bundle TM on a manifold M comes with a continuous map $\pi: TM \rightarrow M$ back down to the manifold, sending $(m, v) \mapsto m$. One might say that π “forgets the tangent vector and just remembers the point it emanated from.” By Example 7.30, π defines a sheaf \mathbf{Sec}_π . It is called the sheaf of *vector fields on M* , and this is what we were describing when we spoke of the sheaf of wind velocities on Earth, above. Given an open subset $U \subseteq M$, an element $v \in \mathbf{Sec}_\pi(U)$ is called a vector field over U because it continuously assigns a tangent vector $v(u)$ to each point $u \in U$. The tangent vector at u tells us the velocity of the wind at that point.

Here's a fun digression: in the case of a spherical manifold M like the Earth, it's possible to prove that for every open set U , as long as $U \neq M$, there is a vector field $v \in \mathbf{Sec}_\pi(U)$ that is never 0: the wind could be blowing throughout U . However, a theorem of Poincaré says that if you look at the whole sphere, there is guaranteed to be a point $m \in M$ at which the wind is not blowing at all. It's like the eye of a hurricane or perhaps a cowlick. A cowlick in someone's hair occurs when the hair has no direction to go, so it sticks up! Hair sticking up would not count as a tangent

vector: tangent vectors must start out lying flat along the head. Poincaré proved that if your head was covered completely with inch-long hair, there would be at least one cowlick. This difference between local sections (over arbitrary $U \subseteq X$) and global sections (over X)—namely that hair can be well-combed whenever $U \neq X$ but cannot be well-combed when $U = X$ —can be thought of as a generative effect, and can be measured by cohomology (see Section 1.6). \blacklozenge

Example 7.32. For every topological space (X, \mathbf{Op}) , we have the topos of sheaves on it. The topos of sets, which one can regard as the story of set theory, is the category of sheaves on the one-point space $\{*\}$. All of that is just to say that the category of sets is the topos of sheaves on a single point. Imagine how much more complex arbitrary toposes are, when they can take place on much more interesting topological spaces (and in fact even more general “sites”). \blacklozenge

Exercise 7.33. Consider the Sierpinski space $(\{1, 2\}, \mathbf{Op}_1)$ from Example 7.19.

1. What is the category \mathbf{Op} for this space? (You may have already figured this out in Exercise 7.20; if not, do so now.)
2. What is a presheaf on \mathbf{Op} ?
3. What is the sheaf condition for \mathbf{Op} ?
4. How do we identify a sheaf on \mathbf{Op} with a function? \blacklozenge

7.4 Toposes

A *topos* is defined to be a category of sheaves.⁸ So for any topological space (X, \mathbf{Op}) , the category $\mathbf{Shv}(X, \mathbf{Op})$ defined in Definition 7.24 is a topos. In particular, taking $X = 1$, the category \mathbf{Set} is a topos, as we’ve been saying all along and saw again explicitly in Example 7.32. And for any database schema—i.e. finitely presented category— C , the category $C\text{-Inst}$ of database instances on C is also a topos.⁹ Toposes encompass both of these sources of examples, and many more.

Toposes are incredibly nice structures, for a variety of seemingly disparate reasons. In this sketch, the reason in focus is that every topos has many of the same structural properties that the category \mathbf{Set} has. Indeed, we discussed in Section 7.2.1 that every topos has limits and colimits, is cartesian closed, has epi-mono factorizations, and has a subobject classifier (see Section 7.2.2). Using these properties, one can do logic with semantics in the topos \mathcal{E} . We explained this for sets, but now imagine it for sheaves on a topological space. There, the same logical symbols $\wedge, \vee, \neg, \Rightarrow, \exists, \forall$ become operations that mean something about sub-sheaves—e.g. vector fields, sections of continuous functions, etc.—not just subsets.

⁸This is sometimes called a *sheaf topos* or a *Grothendieck topos*. There is a more general sort of topos called an *elementary topos* due to Lawvere.

⁹We said that a topos is a category of sheaves, yet database instances are presheaves; so how is $C\text{-Inst}$ a topos? Well, presheaves in fact count as sheaves. We apologize that this couldn’t be clearer. All of this could be made formal if we were to introduce *sites*. Unfortunately, that concept is simply too abstract for the scope of this chapter.

To understand this more deeply, we should say what the subobject classifier $\text{true}: 1 \rightarrow \Omega$ is in more generality. We said that the object Ω of the subobject classifier in the topos \mathbf{Set} is the booleans \mathbb{B} . In a sheaf topos $\mathcal{E} = \mathbf{Shv}(X, \mathbf{Op})$, the object $\Omega \in \mathcal{E}$ is a sheaf, not just a set. What sheaf is it?

7.4.1 The subobject classifier Ω in a sheaf topos

In this subsection we aim to understand the subobject classifier Ω , i.e. the object of truth values, in the sheaf topos $\mathbf{Shv}(X, \mathbf{Op})$. Since Ω is a sheaf, let's understand it by going through the definition of sheaf (Definition 7.24) slowly in this case. A sheaf Ω is a presheaf that satisfies the sheaf condition. As a presheaf it is just a functor $\Omega: \mathbf{Op}^{\text{op}} \rightarrow \mathbf{Set}$; it assigns a set $\Omega(U)$ to each open $U \subseteq X$ and comes with a restriction map $\Omega(U) \rightarrow \Omega(V)$ whenever $V \subseteq U$. So in our quest to understand Ω , we first ask the question: what presheaf is it?

The answer to our question is that Ω is the presheaf that assigns to $U \in \mathbf{Op}$ the set of open subsets of U :

$$\Omega(U) := \{U' \in \mathbf{Op} \mid U' \subseteq U\}. \quad (7.13)$$

That was easy, right? And given $V \subseteq U$, the restriction function $\Omega(U) \rightarrow \Omega(V)$ sends U' to $U' \cap V$. One can check that this is functorial—see Exercise 7.34—and after doing so we will still need to see that it satisfies the sheaf condition. But at least we don't have to struggle to understand Ω : it's a lot like \mathbf{Op} itself.

Exercise 7.34.

1. Show that the definition of Ω given above is functorial, i.e., that whenever $U \subseteq V \subseteq W$, the restriction map $\Omega(W) \rightarrow \Omega(V)$ followed by the restriction map $\Omega(V) \rightarrow \Omega(U)$ is the same as the restriction map $\Omega(W) \rightarrow \Omega(U)$.
2. Is that all that's necessary to conclude that Ω is a presheaf? ◇

To see that Ω as defined in Eq. (7.13) satisfies the sheaf condition (see Definition 7.24), suppose that we have a cover $U = \bigcup_{i \in I} U_i$, and suppose given an element $V_i \in \Omega(U_i)$, i.e. an open set $V_i \subseteq U_i$, for each $i \in I$. Suppose further that for all $i, j \in I$, it is the case that $V_i \cap U_j = V_j \cap U_i$, i.e. that the elements form a matching family. Define $V := \bigcup_{i \in I} V_i$; it is an open subset of U , so we can consider V as an element of $\Omega(U)$. The following verifies that V is a gluing for the $(V_i)_{i \in I}$:

$$V \cap U_j = \left(\bigcup_{i \in I} V_i \right) \cap U_j = \bigcup_{i \in I} (V_i \cap U_j) = \bigcup_{i \in I} (V_j \cap U_i) = \left(\bigcup_{i \in I} U_i \right) \cap V_j = V_j$$

In other words $V \cap U_j = V_j$ for any $j \in I$.

The eagle-eyed reader will have noticed that we haven't yet said all the data needed to define a subobject classifier. To turn the object Ω into the data of a subobject classifier, we also need to give a sheaf morphism $\text{true}: 1 \rightarrow \Omega$. Here $1: \mathbf{Op}^{\text{op}} \rightarrow \mathbf{Set}$ is the terminal sheaf; it maps every open set to the terminal, one element set 1 . The morphism $\text{true}: 1 \rightarrow \Omega$ then is then the sheaf morphism that assigns, for every $U \in \mathbf{Op}$ the function that takes the unique element of 1 to the open set $U \in \Omega(U)$.

Upshot: Truth values are open sets The point is that the truth values in the topos of sheaves on a space (X, \mathbf{Op}) are the open sets of that space. When someone says “is property P true?”, the answer is not yes or no, but “it is true on the open subset U ”. If this U is everything, $U = X$, then P is really true; if U is nothing, $U = \emptyset$, then P is really false. But in general, it’s just true some places and not others.

Example 7.35. If $X = \{1\}$ is a one-point space, we already know three things:

1. $\mathbf{Shv}(X) = \mathbf{Set}$ by Example 7.32.
2. The subobject classifier for \mathbf{Set} is $\Omega = \mathbb{B} = \{\text{true}, \text{false}\}$ by Eq. (7.5).
3. Ω is the sheaf of open sets, by Eq. (7.13).

To justify 2 and 3, it suffices to recognize that X has exactly two open sets, namely $\{1\}$ and \emptyset , and these correspond to the truth values true and false respectively. \blacklozenge

7.4.2 Logic in a sheaf topos

Let’s consider the logical connectives, AND, OR, IMPLIES, and NOT. Suppose we have an open set $U \in \mathbf{Op}$. Given two open sets V_1, V_2 , considered as truth values $V_1, V_2 \in \Omega(U)$, then their conjunction “ U AND V ” is their intersection, and their disjunction “ U OR V ” is their union;

$$(U \wedge V) := U \cap V \quad \text{and} \quad (U \vee V) := U \cup V. \quad (7.14)$$

These formulas are easy to remember, because \wedge looks like \cap and \vee looks like \cup . The implication $U \Rightarrow V$ is the largest open set R such that $R \cap U \subseteq V$, i.e.

$$(U \Rightarrow V) := \bigcup_{\{R \in \mathbf{Op} \mid R \cap U \subseteq V\}} R. \quad (7.15)$$

In general, it is not easy to reduce this further, so implication is the hardest logical connective to think about topologically.

Finally, the negation of U is given by $\neg U := (U \Rightarrow \text{false})$, and this turns out to be relatively simple. By the formula in Eq. (7.15), it is the union of all R such that $R \cap U = \emptyset$, i.e. the union of all open sets in the complement of U . In other words, $\neg U$ is the interior of the complement of U .

Example 7.36. Consider the real line $X = \mathbb{R}$ as a topological space (see Exercise 7.16). Let $U, V \in \Omega(X)$ be the open sets $U = \{x \in \mathbb{R} \mid x < 3\}$ and $V = \{x \in \mathbb{R} \mid -4 < x < 4\}$. Using interval notation, $U = (-\infty, 3)$ and $V = (-4, 4)$. Then

- $U \wedge V = (-4, 3)$.
- $U \vee V = (-\infty, 4)$.
- $\neg U = (3, \infty)$.
- $\neg V = (-\infty, -4) \cup (4, \infty)$.
- $(U \Rightarrow V) = (-4, \infty)$
- $(V \Rightarrow U) = U$

\blacklozenge

Exercise 7.37. Consider the real line \mathbb{R} as a topological space, and consider the open subset $U = \mathbb{R} - \{0\}$.

1. What open subset is $\neg U$?
2. What open subset is $\neg\neg U$?
3. Is it true that $U \subseteq \neg\neg U$?
4. Is it true that $\neg\neg U \subseteq U$?

◇

Above we explained operations on open sets, one corresponding to each logical connective, but there are also open sets corresponding to the symbols \top and \perp , which mean true and false.

- Exercise 7.38.*
1. The symbol \top corresponds to an open set $U \in \mathbf{Op}$ such that for any open set $V \in \mathbf{Op}$, we have $(\top \wedge V) = V$. Which open set is it?
 2. Other things we should expect from \top include $(\top \vee V) = \top$ and $(V \Rightarrow \top) = \top$ and $(\top \Rightarrow V) = V$. Do these hold for your answer to 1?
 3. The symbol \perp corresponds to an open set $U \in \mathbf{Op}$ such that for any open set $V \in \mathbf{Op}$, we have $(\perp \vee V) = V$. Which open set is it?
 4. Other things we should expect from \perp include $(\perp \wedge V) = \perp$ and $(\perp \Rightarrow V) = \top$. Do these hold for your answer to 1?

◇

7.4.3 Predicates

In English, a predicate is the part of the sentence that comes after the subject. For example “...is even” or “...likes the weather” are predicates. Not every subject makes sense for a given predicate; e.g. the sentence “7 is even” may be false, but it makes sense. In contrast, the sentence “2.7 is even” does not really make sense, and “2.7 likes the weather” certainly doesn’t. In computer science, they might say “The expression ‘2.7 likes the weather’ does not type check.”

The point is that each predicate is associated to a type, namely the type of subject that makes sense for that predicate. When we apply a predicate to a subject of the appropriate type, the result has a truth value: “7 is even” is either true or false. Perhaps “Bob likes the weather” is true some days and false on others. In fact, this truth value might change by the year (bad weather this year), by the season, by the hour, etc. In English, we expect truth values of sentences to change over time, which is exactly the motivation for this chapter. We’re working toward a logic where truth values change over time.

In a topos $\mathcal{E} = \mathbf{Shv}(X, \mathbf{Op})$, a predicate is a sheaf morphism $p: S \rightarrow \Omega$ where $S \in \mathcal{E}$ is a sheaf and $\Omega \in \mathcal{E}$ is the subobject classifier, the sheaf of truth values. By Definition 7.24 we get a function $p(U): S(U) \rightarrow \Omega(U)$ for any open set $U \subseteq X$. In the above example—which we will discuss more carefully in Section 7.5—if S is the sheaf of people (people come and go over time), and $\text{Bob} \in S(U)$ is a person existing over a time U , and p is “likes the weather”, then $p(\text{Bob}) \subseteq U$ is the subset of times when Bob likes the weather. So the answer to “Bob likes the weather” is something like “in summers yes, and also in April 2018 and May 2019 yes, but in all other times no”. That’s $p(\text{Bob})$, the temporal truth value obtained by applying the predicate p to the subject Bob.

Exercise 7.39. Just now we described how a predicate $p: S \rightarrow \Omega$, such as “. . . likes the weather”, acts on sections $s \in S(U)$, say $s = \text{Bob}$. But by Definition 7.6, any predicate $p: S \rightarrow \Omega$ to the subobject classifier also defines a subobject of $\{S \mid p\} \subseteq S$. Describe the sections of this subsheaf. \diamond

The poset of subobjects For any topos $\mathcal{E} = \mathbf{Shv}(X, \mathbf{Op})$ and object (sheaf) $S \in \mathcal{E}$, the set of predicates $\mathcal{E}(S, \Omega)$ forms a poset, which we denote

$$(|\Omega^S|, \leq^S) \quad (7.16)$$

The elements of $|\Omega^S|$ are the predicates on S ; the order relation \leq^S is as follows. Given two predicates $p, q: S \rightarrow \Omega$, we say that $p \leq^S q$ if the first implies the second. More precisely, for any $U \in \mathbf{Op}$ and section $s \in S(U)$ we obtain two open subsets $p(s) \subseteq U$ and $q(s) \subseteq U$. We say that $p \leq^S q$ if $p(s) \subseteq q(s)$ for all $U \in \mathbf{Op}$ and $s \in S(U)$. We often drop the superscript from \leq^S and simply write \leq . In formal logic notation, one might write $p \leq^S q$ using the \vdash symbol, e.g. in one of the following ways:

$$s : S \mid p(s) \vdash q(s) \quad \text{or} \quad p(s) \vdash_{s:S} q(s).$$

In particular, if $S = 1$ is the terminal object, we denote $|\Omega^S|$ by $|\Omega|$, and refer to elements $p \in |\Omega|$ as *propositions*. They are just morphisms $p: 1 \rightarrow \Omega$.

All of the logical symbols ($\top, \perp, \wedge, \vee, \Rightarrow, \neg$) from Section 7.4.2 make sense in any such poset $|\Omega^S|$. For any two predicates $p, q: S \rightarrow \Omega$, we define $(p \wedge q): S \rightarrow \Omega$ by $(p \wedge q)(s) := p(s) \wedge q(s)$, and similarly for \vee . Thus one says that these operations are *computed pointwise* on S . With these definitions, the \wedge symbol is the meet and the \vee symbol is the join—in the sense of Definition 1.60—for the poset $|\Omega^S|$.

With all of the logical structure we’ve defined so far, the poset $|\Omega^S|$ of predicates on S forms what’s called a Heyting algebra. We will not define it here, but more information can be found in Section 7.6. We now move on to quantification.

7.4.4 Quantification

Quantification comes in two flavors: universal and existential, or “for all” and “there exists”. Each takes in a predicate of $n+1$ variables and returns a predicate of n variables.

Example 7.40. Suppose we have two sheaves $S, T \in \mathbf{Shv}(X, \mathbf{Op})$ and a predicate $p: S \times T \rightarrow \Omega$. Let’s say T represents what’s considered newsworthy and S is again the set of people. So for a subset of time U , a section $t \in T(U)$ means a something that’s considered newsworthy throughout the whole of U , and a section $s \in S(U)$ means a person that lasts throughout the whole of U . Let’s imagine the predicate p as “ s is worried about t ”. Now recall from Section 7.4.3 that a predicate p does not simply return true or false; given a person s and a news-item t , it returns a truth value corresponding to the subset of times on which $p(s, t)$ is true.

“For all t in T , ... is worried about t ” is itself a predicate on just one variable, S , which we denote

$$\forall(t : T). p(s, t).$$

Applying this predicate to a person s returns the times when that person is worried about everything in the news. Similarly, “there exists t in T such that s is worried about t ” is also a predicate on S , which we denote $\exists(t : T). p(s, t)$. If we apply this predicate to a person s , we get the times when person s is worried about at least one thing in the news. ♦

Exercise 7.41. In the topos **Set**, where $\Omega = \mathbb{B}$, consider the predicate $p : \mathbb{N} \times \mathbb{Z} \rightarrow \mathbb{B}$ given by

$$p(n, z) = \begin{cases} \text{true} & \text{if } n \leq |z| \\ \text{false} & \text{if } n > |z|. \end{cases}$$

1. What is the set of $n \in \mathbb{N}$ for which the predicate $\forall(z : \mathbb{Z}). p(n, z)$ holds?
2. What is the set of $n \in \mathbb{N}$ for which the predicate $\exists(z : \mathbb{Z}). p(n, z)$ holds?
3. What is the set of $z \in \mathbb{Z}$ for which the predicate $\forall(n : \mathbb{N}). p(n, z)$ holds?
4. What is the set of $z \in \mathbb{Z}$ for which the predicate $\exists(n : \mathbb{N}). p(n, z)$ holds?

♦

So given p , we have universally- and existentially-quantified predicates $\forall(t : T). p(s, t)$ and $\exists(t : T). p(s, t)$ on S . How do we formally understand them as sheaf morphisms $S \rightarrow \Omega$ or, equivalently, as subsheaves of S ?

Universal quantification Given a predicate $p : S \times T \rightarrow \Omega$, the universally-quantified predicate $\forall(t : T). p(s, t)$ takes a section $s \in S(U)$, for any open set U , and returns a certain open set $V \in \Omega(U)$. Namely, it returns the largest open set V for which $p(s|_V, t) = V$ for all $t \in T(V)$.

Exercise 7.42. Apply the above definition to the “person s is worried about news t ” example above.

1. What open set is $\forall(t : T). p(s, t)$ for a person s ?
2. Does it have the semantic meaning you’d expect, given the less formal description in Section 7.4.4? ♦

Abstractly speaking, the universally-quantified predicate corresponds to the subsheaf given by the following pullback:

$$\begin{array}{ccc} \forall_t p & \longrightarrow & 1 \\ \downarrow & \lrcorner & \downarrow \text{true}^T \\ S & \xrightarrow{p'} & \Omega^T \end{array}$$

where $p' : S \rightarrow \Omega^T$ is the currying of $S \times T \rightarrow \Omega$ and true^T is the currying of the composite $1 \times T \xrightarrow{!} 1 \xrightarrow{\text{true}} \Omega$. See Eq. (7.3).

Existential quantification Given a predicate $p: S \times T \rightarrow \Omega$, the existentially quantified predicate $\exists(t: T). p(s, t)$ takes a section $s \in S(U)$, for any open set U , and returns a certain open set $V \in \Omega(U)$. To find it, consider all the open sets V_i for which there exists some $t_i \in T(V_i)$ satisfying $p(s|_{V_i}, t_i) = V_i$; then $V := \bigcup_i V_i$ is their union. Note that there may be no $t \in T(V)$ such that $p(s, t)$ holds! Thus the existential quantifier is doing a lot of work “under the hood”, taking into account the idea of coverings.

Exercise 7.43. Apply the above definition to the “person s is worried about news t ” example above.

1. What open set is $\exists(t: T). p(s, t)$ for a person s ?
2. Does it have the semantic meaning you’d expect? ◇

Abstractly speaking, the existentially-quantified predicate is given as follows. Start with the subobject classified by p , namely $\{(s, t) \in S \times T \mid p(s, t)\} \subseteq S \times T$, compose with the projection $\pi_S: S \times T \rightarrow S$ as on the upper right; then take the epi-mono factorization as on the lower left:

$$\begin{array}{ccc} \{S \times T \mid p\} & \xrightarrow{\quad} & S \times T \\ \downarrow & & \downarrow \pi_S \\ \exists_t p & \xrightarrow{\quad} & S \end{array}$$

Then the bottom map is the desired subsheaf.

7.4.5 Modalities

Back in Example 1.95 we discussed modal operators—also known as modalities—saying they are closure operators on posets which arise in logic. The posets we were referring to are the ones discussed in Eq. (7.16): for any object $S \in \mathcal{E}$ there is the poset $(|\Omega^S|, \leq^S)$ of predicates on S , where $|\Omega^S| = \mathcal{E}(S, \Omega)$ is just the set of morphisms $S \rightarrow \Omega$ in the category \mathcal{E} .

Definition 7.44. A *modality* in \mathcal{E} is a sheaf morphism $j: \Omega \rightarrow \Omega$ satisfying three properties:

- (a) $\text{id}_\Omega \leq^\Omega j$, where $\text{id}_\Omega: \Omega \rightarrow \Omega$ is the identity;
- (b) $j.j \leq j$; and
- (c) the following diagram commutes:

$$\begin{array}{ccc} \Omega \times \Omega & \xrightarrow{\wedge} & \Omega \\ j \times j \downarrow & & \downarrow j \\ \Omega \times \Omega & \xrightarrow{j} & \Omega \end{array}$$

In Example 1.95 we informally said that for any proposition p , e.g. “Bob is in San Diego”, there is a modal operator “assuming p , ...”. Now we are in a position to make that formal.

Proposition 7.45. Fix a proposition $p \in |\Omega|$. Then

1. the sheaf morphism $\Omega \rightarrow \Omega$ given by sending q to $p \Rightarrow q$ is a modality.
2. the sheaf morphism $\Omega \rightarrow \Omega$ given by sending q to $p \vee q$ is a modality.
3. the sheaf morphism $\Omega \rightarrow \Omega$ given by sending q to $(q \Rightarrow p) \Rightarrow p$ is a modality.

We cannot prove Proposition 7.45 here, but we give references in Section 7.6.

7.4.6 Type theories and semantics

We have been talking about the logic of a topos in terms of open sets, but this is actually a conflation of two ideas that are really better left unconfliated. The first is logic, or formal language, and the second is semantics, or meaning. The formal language looks like this:

$$\forall(t : T). \exists(s : S). f(s) = t$$

and semantic statements are like “the sheaf morphism $f : S \rightarrow T$ is an epimorphism”. In the former, logical world, all statements are formed according to strict rules and all proofs are deductions that also follow strict rules. In the latter, semantic world, statements and proofs are about the sheaves themselves, as mathematical objects. We admit these are rough statements; again, our aim here is only an invitation to further reading.

To *provide semantics* for a logical system means to provide a translation system for converting all logical statements in a formal language into mathematical statements about particular sheaves and their relationships. A computer can carry out logical deductions without knowing what any of them “mean” about sheaves. We say that semantics is *sound* if every formal proof is converted into a true fact about the relevant sheaves.

Every topos can be assigned a formal language, often called an *internal language*, in which to carry out constructions and formal proofs. This language has a sound semantics, which goes under the name *categorical semantics* or *Kripke-Joyal semantics*. We gave the basic ideas in Section 7.4; we give references to the literature in Section 7.6.

7.5 A topos of behavior types

Now that we have discussed logic in a sheaf topos, we return to our motivating example, a topos of behavior types. We begin by discussing the topological space on which behavior types will be sheaves, a space called the *interval domain*.

7.5.1 The interval domain

The interval domain \mathbb{IR} is a specific topological space, which we will use to model intervals of time. In other words, we will be interested in the category $\mathbf{Shv}(\mathbb{IR})$ of sheaves on the interval domain.

To give a topological space, one must give a pair (X, \mathbf{Op}) , where X is a set “of points” and \mathbf{Op} is a topology on X ; see Definition 7.14. The set of points for $\mathbb{I}\mathbb{R}$ is that of all finite closed intervals

$$\mathbb{I}\mathbb{R} := \{[d, u] \subseteq \mathbb{R} \mid d \leq u\}.$$

For $a < b$ in \mathbb{R} , let $o_{[a,b]}$ denote the set $\{[d, u] \in \mathbb{I}\mathbb{R} \mid a < d \leq u < b\}$; these are called *basic open sets*. The topology \mathbf{Op} is determined by these basic open sets in that a subset U is open if it is the union of some collection of basic open sets.

Thus for example, $o_{[0,5]}$ is an open set: it contains every $[d, u]$ contained in the open interval $\{x \in \mathbb{R} \mid 0 < x < 5\}$. Similarly $o_{[4,8]}$ is an open set, but note that $o_{[0,5]} \cup o_{[4,8]} \neq o_{[0,8]}$. Indeed, the interval $[2, 6]$ is in the right-hand side but not the left.

Exercise 7.46.

1. Explain why $[2, 6] \in o_{[0,8]}$.
2. Explain why $[2, 6] \notin o_{[0,5]} \cup o_{[4,8]}$. ◇

Let \mathbf{Op} denote the open sets of $\mathbb{I}\mathbb{R}$, as described above, and let $\mathbf{BT} = \mathbf{Shv}(\mathbb{I}\mathbb{R}, \mathbf{Op})$ denote the topos of sheaves on this space. We call it the topos of *behavior types*.

There is an important subspace of $\mathbb{I}\mathbb{R}$, namely the usual space of real numbers \mathbb{R} . We see \mathbb{R} as a subspace of $\mathbb{I}\mathbb{R}$ via the isomorphism

$$\mathbb{R} \cong \{[d, u] \in \mathbb{I}\mathbb{R} \mid d = u\}.$$

We discussed the usual topology on \mathbb{R} in Example 7.15, but we also get a topology on \mathbb{R} because it is a subset of $\mathbb{I}\mathbb{R}$; i.e. we have the subspace topology as described in Exercise 7.21. These agree, as the reader can check.

Exercise 7.47. Show that a subset $U \subseteq \mathbb{R} \subseteq \mathbb{I}\mathbb{R}$ is open, i.e. that $U \in \mathbf{Op}$, iff $U \cap \mathbb{R}$ is open in the usual topology on \mathbb{R} defined in Example 7.15. ◇

7.5.2 Sheaves on $\mathbb{I}\mathbb{R}$

We cannot go into much depth about the sheaf topos $\mathbf{BT} = \mathbf{Shv}(\mathbb{I}\mathbb{R}, \mathbf{Op})$, for reasons of space; we refer the interested reader to Section 7.6. In this section we will briefly discuss what it means to be a sheaf on $\mathbb{I}\mathbb{R}$, giving a few examples including that of the subobject classifier.

What is a sheaf on $\mathbb{I}\mathbb{R}$? A sheaf S on the interval domain $(\mathbb{I}\mathbb{R}, \mathbf{Op})$ is a functor $S: \mathbf{Op}^{\text{op}} \rightarrow \mathbf{Set}$: it assigns to each open set U a set $S(U)$; how should we interpret this? An element $s \in S(U)$ is something that S says is an “event that takes place throughout the interval U ”. Given this U -event s together with an open subset $V \subseteq U$, there is a V -event $s|_V$ that tells us what s is doing throughout V . If $U = \bigcup_{i \in I} U_i$ and we can find matching U_i -events (s_i) for each $i \in I$, then the sheaf condition (Definition 7.24) says that there is a unique U -event $s \in S(U)$ that encompasses all of them: $s|_{U_i} = s_i$ for each $i \in I$.

We said in Section 7.5.1 that every open set $U \subseteq \mathbb{R}$ can be written as the union of basic open sets $o_{[a,b]}$. This implies that any sheaf S is determined by its values $S(o_{[a,b]})$ on these basic open sets. The sheaf condition furthermore implies that these vary continuously in a certain sense, which we can express formally as

$$S(o_{[a,b]}) \cong \lim_{\epsilon > 0} S(o_{[a-\epsilon, a+\epsilon]}).$$

However, rather than get into the details, we describe a few sorts of sheaves that may be of interest.

Example 7.48. For any set A there is a sheaf $\mathbf{A} \in \mathbf{Shv}(\mathbb{R})$ that assigns to each open set U the set $\mathbf{A}(U) := A$. This allows us to refer to integers, or real numbers, or letters of an alphabet, as though they were behaviors. What sort of behavior is $7 \in \mathbb{N}$? It is the sort of behavior that never changes: it's always seven. Thus \mathbf{A} is called the *constant sheaf on A* . \blacklozenge

Example 7.49. Fix any topological space (X, \mathbf{Op}_X) . Then there is a sheaf F_X of *local functions from \mathbb{R} to X* . That is, for any open set $U \in \mathbf{Op}_{\mathbb{R}}$, we assign the set $F_X(U) := \{f: U \rightarrow X \mid f \text{ is continuous}\}$. There is also the sheaf G_X of local functions on the subspace $\mathbb{R} \subseteq \mathbb{R}$. That is, for any open set $U \in \mathbf{Op}_{\mathbb{R}}$, we assign the set $G_X(U) := \{f: U \cap \mathbb{R} \rightarrow X \mid f \text{ is continuous}\}$. \blacklozenge

Exercise 7.50. Fix any topological space (X, \mathbf{Op}_X) and any subset $R \subseteq \mathbb{R}$ of the interval domain. Define $H_X(U) := \{f: U \cap R \rightarrow X \mid f \text{ is continuous}\}$.

1. Is H_X a presheaf? If not, why not; if so, what are the restriction maps?
2. Is H_X a sheaf? Why or why not? \blacklozenge

Example 7.51. Another source of examples comes from the world of open hybrid dynamical systems. These are machines whose behavior is a mixture of continuous movements—generally imagined as trajectories through a vector field—and discrete jumps. These jumps are imagined as being caused by signals that spontaneously arrive. Over any interval of time, a hybrid system has certain things that it can do and certain things that it cannot. Although we will not make this precise here, there is a construction for converting any hybrid system into a sheaf on \mathbb{R} ; we will give references in Section 7.6. \blacklozenge

We refer to sheaves on \mathbb{R} as behavior types because almost any sort of behavior one can imagine is a behavior type. Of course, a complex behavior type—such as the way someone acts when they are in love—would be extremely hard to write down. But the idea is straightforward: for any interval of time, say a three-day interval $(d, d + 3)$, let $L(d, d + 3)$ denote the set of all possible behaviors a person who is in love could possibly do. Obviously it's a big, unwieldy set, and not one someone would want to make precise. But to the extent that one can imagine that sort of behavior as occurring through time, they could imagine the corresponding sheaf.

The subobject classifier as a sheaf on \mathbb{R} In any sheaf topos, the subobject classifier Ω is itself a sheaf. It is responsible for the truth values in the topos. As we said in Section 7.4.1, when it comes to sheaves on a topological space (X, \mathbf{Op}) , truth values are open subsets $U \in \mathbf{Op}$.

BT is the topos of sheaves on the space $(\mathbb{R}, \mathbf{Op})$, as defined in Section 7.5.1. As always, the subobject classifier Ω assigns to any $U \in \mathbf{Op}$ the set of open subsets of U , so these are the truth values. But what do they mean? The idea is that every proposition, such as “Bob likes the weather” returns an open set U , as if to respond that Bob likes the weather “...throughout time period U ”. Let’s explore this just a bit more.

Suppose Bob likes the weather throughout the interval $(0, 5)$ and throughout the interval $(4, 8)$. We would probably conclude that Bob likes the weather throughout the interval $(0, 8)$. But what about the more ominous statement “a single pair of eyes has remained watching position p ”. Then just because it’s true on $(0, 5)$ and on $(4, 8)$, does not imply that it’s been true on $(0, 8)$: there may have been a change of shift, where one watcher was relieved from their post by another watcher. As another example, consider the statement “the stock market did not go down by more than 10 points”. This might be true on $(0, 5)$ and true on $(4, 8)$ but not on $(0, 8)$. In order to capture the semantics of statements like these—statements that take time to evaluate—we must use the space \mathbb{R} rather than the space \mathbb{R} .

7.5.3 Safety proofs in temporal logic

We now have at least a basic idea of what goes into a proof of safety, say for airplanes in the national airspace system (NAS). The NAS consists of many different systems interacting: interactions between airplanes, each of which is an interaction between physics, humans, sensors, and actuators, each of which is an interaction between still more basic parts. Suppose that each of the systems—at any level—is guaranteed to satisfy some property. For example, perhaps we can assume that an airplane engine is either out of gas, has a broken fuel line, or is following the orders of the pilot. Of course, this is just a model, but all human reasoning is via models so this is the best we can hope for.

Thus we assume that we have some basic systems which are guaranteed to have certain behavior. If there is a rupture in the fuel line, the sensors will alert the pilot within 10 seconds, etc. Each of the components interact with a number of different variables: a pilot interacts with the radio, the positions of the dials, the position of the thruster, and the visual data in front of her. The component—here the pilot—is guaranteed to keep these variables in some relation: “if I see something, I will say something” or “if the dials are in position *bad_pos*, I will engage the thruster within 1 second”. We call these guarantees *behavior contracts*.

All of the above can be captured in the topos **BT** of behavior types. The variables are behavior types: the altimeter is a variable whose value $\theta \in \mathbb{R}_{\geq 0}$ is changing

continuously with respect to time. The thruster is also a continuously-changing variable whose value is in the range $[0, 1]$, etc.

The guaranteed relationships—behavior contracts—are given by predicates on variables. For example, if the pilot will always engage the thruster within one second of the dials being in position `bad_pos`, this can be captured by a predicate $p: \text{dials} \times \text{thrusters} \rightarrow \Omega$. While we have not written out a formal language for p , one could imagine $p(D, T)$ for $D: \text{dials}$ and $T: \text{thrusters}$ as

$$\begin{aligned} \forall(t: \mathbb{R}). @_t(\text{bad_pos}(D)) \Rightarrow \\ \exists(r: \mathbb{R}). (0 < r < 1) \wedge \forall(r': \mathbb{R}). 0 \leq r' \leq 1 \Rightarrow @_{t+r+r'}(\text{engaged}(T)). \end{aligned} \quad (7.17)$$

Here $@_t$ is a modality, as we discussed in Definition 7.44; in fact it turns out to be one of type 3. from Proposition 7.45, but we cannot go into that. For a proposition q , the statement $@_t(q)$ says that q is true in some small enough neighborhood around t . So (7.17) says “for all times t , if in a small enough neighborhood around t the dials are in a bad position, then within one second, the thrusters will be engaged for at least one second.”

Given an actual playing-out-of-events over a time period U , i.e. actual section $D \in \text{dials}(U)$ and $T \in \text{thrusters}(U)$, the predicate Eq. (7.17) will hold on certain parts of U and not others, and this is the truth value of p . Hopefully the pilot upholds her behavior contract at all times she is flying, in which case the truth value will be true throughout that interval U . But if the pilot breaks her contract over certain intervals, then this fact is recorded in Ω .

The logic allows us to record such axioms and then reason from them: e.g. if the pilot and the airplane, and at least one of the three radars upholds its contract then safe separation will be maintained. We cannot give further details here, but these matters have been worked out in detail in [SS17]; see Section 7.6.

7.6 Summary and further reading

This chapter was about modeling various sorts of behavior using sheaves on a space of time-intervals. Behavior may seem like it’s something that occurs now in the present, but in fact our memory of past behavior informs what the current behavior means. In order to commit to anything, to plan or complete any sort of process, one needs to be able to reason over time-intervals. The nice thing about temporal sheaves—indeed sheaves on any site—is that they fit into a categorical structure called a topos, which has many useful formal properties. In particular, it comes equipped with a higher-order logic with which we can formally reason about how temporal sheaves work together when combined in larger systems. A much more detailed version of this story was presented in [SS17]. But it would have been impossible without the extensive edifice of topos theory and domain theory that has been developed over the past six decades.

Sheaves and toposes were invented by Grothendieck and his school in the 1960s [AGV71] as an approach to proving conjectures at the intersection of algebraic geometry and number theory, called the Weil conjectures. Soon after, Lawvere and Tierney recognized that toposes had all the structure necessary to do logic, and with a whole host of other category theorists, the subject was developed to an impressive extent in many directions. For a much more complete history, see [McL90].

There are many sorts of references on topos theory. One that starts by introducing categories and then moves to toposes, focusing on logic, is [McL92]. Our favorite treatment is perhaps [MM92], where the geometric aspects play a central role. Finally, Johnstone has done the field a huge favor by collecting large amounts of the theory into a single two-volume set [Joh02]; it is very dense, but essential for the serious student. For just categorical (Kripke-Joyal) semantics of logic in a topos, one should see either [MM92], [Jac99], or [LS88].

We did not mention domain theory much in this chapter, aside from referring to the interval domain. But domains, in the sense of Dana Scott, play an important role in the deeper aspects of the theory. A good reference is [Gie+03], but for an introduction we suggest [AJ94].

In some sense our application area has been a very general sort of dynamical system. Other categorical approaches to this subject include [JNW96], [HTP03], [AS05], and [Law86], though there are many others.

We hope you have enjoyed the seven sketches in this book. As a next step, consider running a reading course on applied category theory with some friends or colleagues. Simultaneously, we hope you begin to search out categorical ways of thinking about familiar subjects. Perhaps you'll find something you want to contribute to this growing field of applied category theory, or as we sometimes call it, the field of compositionality.

Bibliography

- [Ada17] Elie M. Adam. “Systems, Generativity and Interactional Effects”. available online: [eliadamthesis](#). PhD thesis. Massachusetts Institute of Technology, July 2017.
- [AGV71] Michael Artin, Alexander Grothendieck, and Jean-Louis Verdier. *Theorie de Topos et Cohomologie Etale des Schemas I, II, III*. Vol. 269, 270, 305. Lecture Notes in Mathematics. Springer, 1971.
- [AJ94] Samson Abramsky and Achim Jung. “Domain theory”. In: *Handbook of logic in computer science*. Oxford University Press. 1994.
- [AS05] Aaron D Ames and Shankar Sastry. “Characterization of Zeno behavior in hybrid systems using homological methods”. In: *American Control Conference, 2005. Proceedings of the 2005*. IEEE. 2005, pp. 1160–1165.
- [AV93] Samson Abramsky and Steven Vickers. “Quantales, observational logic and process semantics”. In: *Mathematical Structures in Computer Science 3.2* (1993), pp. 161–227.
- [Awo10] Steve Awodey. *Category theory*. Second. Vol. 52. Oxford Logic Guides. Oxford University Press, Oxford, 2010, pp. xvi+311. ISBN: 978-0-19-923718-0.
- [BD98] John C Baez and James Dolan. “Categorification”. In: *arXiv preprint math/9802029* (1998).
- [BE15] John C. Baez and Jason Erbele. “Categories in control”. In: *Theory Appl. Categ.* 30 (2015), Paper No. 24, 836–881. ISSN: 1201-561X.
- [BF15] John C Baez and Brendan Fong. “A compositional framework for passive linear networks”. In: *arXiv preprint arXiv:1504.05625* (2015).
- [BFP16] John C Baez, Brendan Fong, and Blake S Pollard. “A compositional framework for Markov processes”. In: *Journal of Mathematical Physics* 57.3 (2016), p. 033301.
- [BH08] Philip A Bernstein and Laura M Haas. “Information integration in the enterprise”. In: *Communications of the ACM* 51.9 (2008), pp. 72–79.

- [Bor94] Francis Borceux. *Handbook of categorical algebra*. 1. Vol. 50. Encyclopedia of Mathematics and its Applications. Basic category theory. Cambridge University Press, Cambridge, 1994.
- [BP17] John C Baez and Blake S Pollard. “A compositional framework for reaction networks”. In: *Reviews in Mathematical Physics* 29.09 (2017), p. 1750028.
- [BS17] Filippo Bonchi and Fabio Sobociński Paweł and Zanasi. “The calculus of signal flow diagrams I: Linear relations on streams”. In: *Inform. and Comput.* 252 (2017), pp. 2–29. ISSN: 0890-5401. URL: <https://doi.org/10.1016/j.ic.2016.03.002>.
- [BSZ14] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. “A categorical semantics of signal flow graphs”. In: *International Conference on Concurrency Theory*. Springer. 2014, pp. 435–450.
- [BSZ15] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. “Full abstraction for signal flow graphs”. In: *ACM SIGPLAN Notices*. Vol. 50. 1. ACM. 2015, pp. 515–526.
- [BW90] Michael Barr and Charles Wells. *Category theory for computing science*. Vol. 49. Prentice Hall New York, 1990.
- [Car91] Aurelio Carboni. “Matrices, relations, and group representations”. In: *Journal of Algebra* 136.2 (1991), pp. 497–529. ISSN: 0021-8693. DOI: [https://doi.org/10.1016/0021-8693\(91\)90057-F](https://doi.org/10.1016/0021-8693(91)90057-F). URL: <http://www.sciencedirect.com/science/article/pii/002186939190057F>.
- [CD95] Boris Cadish and Zinovy Diskin. “Algebraic graph-based approach to management of multibase systems, I: Schema integration via sketches and equations”. In: *proceedings of Next Generation of Information Technologies and Systems, NGITS*. Vol. 95. 1995.
- [Cen15] Andrea Censi. “A mathematical theory of co-design”. In: *arXiv preprint arXiv:1512.08055* (2015).
- [Cen17] Andrea Censi. “Uncertainty in Monotone Co-Design Problems”. In: *IEEE Robotics and Automation Letters* (Feb. 2017). URL: <https://arxiv.org/abs/1609.03103>.
- [CFS16] Bob Coecke, Tobias Fritz, and Robert W. Spekkens. “A mathematical theory of resources”. In: *Inform. and Comput.* 250 (2016), pp. 59–86. ISSN: 0890-5401. URL: <https://doi.org/10.1016/j.ic.2016.02.008>.
- [con18] Wikipedia contributors. *Symmetric monoidal category* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 22-February-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Symmetric_monoidal_category&oldid=821126747.

- [CW87] A. Carboni and R.F.C. Walters. “Cartesian bicategories I”. In: *Journal of Pure and Applied Algebra* 49.1 (1987), pp. 11–32. ISSN: 0022-4049. DOI: [https://doi.org/10.1016/0022-4049\(87\)90121-6](https://doi.org/10.1016/0022-4049(87)90121-6). URL: <http://www.sciencedirect.com/science/article/pii/0022404987901216>.
- [CY96] Louis Crane and David N Yetter. “Examples of categorification”. In: *arXiv preprint q-alg/9607028* (1996).
- [FGR03] Michael Fleming, Ryan Gunther, and Robert Rosebrugh. “A database of categories”. In: *Journal of Symbolic Computation* 35.2 (2003), pp. 127–135. ISSN: 0747-7171. DOI: [10.1016/S0747-7171\(02\)00104-9](https://doi.org/10.1016/S0747-7171(02)00104-9).
- [Fon15] Brendan Fong. “Decorated cospans”. In: *Theory and Applications of Categories* 30.33 (2015), pp. 1096–1120.
- [Fon16] Brendan Fong. “The Algebra of Open and Interconnected Systems”. PhD thesis. University of Oxford, 2016. URL: <https://arxiv.org/pdf/1609.05382.pdf>.
- [Fon17] Brendan Fong. “Decorated corelations”. In: *arXiv preprint arXiv:1703.09888* (2017).
- [Fra67] John B. Fraleigh. *A first course in abstract algebra*. Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont., 1967, pp. xvi+447.
- [Fri17] Tobias Fritz. “Resource convertibility and ordered commutative monoids”. In: *Math. Structures Comput. Sci.* 27.6 (2017), pp. 850–938. ISSN: 0960-1295. URL: <https://doi.org/10.1017/S0960129515000444>.
- [FSR16] Brendan Fong, Paweł Sobociński, and Paolo Rapisarda. “A categorical approach to open and interconnected dynamical systems”. In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM, 2016, pp. 495–504.
- [Gie+03] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *Continuous lattices and domains*. Vol. 93. Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge, 2003, pp. xxxvi+591. ISBN: 0-521-80338-1. DOI: [10.1017/CB09780511542725](https://doi.org/10.1017/CB09780511542725). URL: <http://dx.doi.org/10.1017/CB09780511542725>.
- [Gla13] K Glazek. *A Guide to the Literature on Semirings and their Applications in Mathematics and Information Sciences: With Complete Bibliography*. Springer Science & Business Media, 2013.
- [HMP98] Claudio Hermida, Michael Makkai, and John Power. “Higher dimensional multigraphs”. In: *Logic in Computer Science, 1998. Proceedings. Thirteenth Annual IEEE Symposium on*. IEEE, 1998, pp. 199–206.
- [HTP03] Efsandiar Haghverdi, Paulo Tabuada, and George Pappas. “Bisimulation relations for dynamical and control systems”. In: *Electronic Notes in Theoretical Computer Science* 69 (2003), pp. 120–136.

- [IP94] Amitavo Islam and Wesley Phoa. “Categorical models of relational databases I: Fibrational formulation, schema integration”. In: *International Symposium on Theoretical Aspects of Computer Software*. Springer. 1994, pp. 618–641.
- [Jac99] Bart Jacobs. *Categorical logic and type theory*. Vol. 141. Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Co., Amsterdam, 1999, pp. xviii+760. ISBN: 0-444-50170-3.
- [JNW96] André Joyal, Mogens Nielsen, and Glynn Winskel. “Bisimulation from open maps”. In: *Information and Computation* 127.2 (1996), pp. 164–185.
- [Joh02] Peter T. Johnstone. *Sketches of an elephant: a topos theory compendium*. Vol. 43. Oxford Logic Guides. New York: The Clarendon Press Oxford University Press, 2002, pp. xxii+468+71. ISBN: 0-19-853425-6.
- [Joh77] P. T. Johnstone. *Topos theory*. London Mathematical Society Monographs, Vol. 10. Academic Press [Harcourt Brace Jovanovich, Publishers], London-New York, 1977, pp. xxiii+367. ISBN: 0-12-387850-0.
- [JR02] Michael Johnson and Robert Rosebrugh. “Sketch Data Models, Relational Schema and Data Specifications”. In: *Electronic Notes in Theoretical Computer Science* 61 (2002). CATS’02, Computing: the Australasian Theory Symposium, pp. 51–63. ISSN: 1571-0661.
- [JS93] André Joyal and Ross Street. “Braided tensor categories”. In: *Adv. Math.* 102.1 (1993), pp. 20–78. ISSN: 0001-8708. URL: <http://dx.doi.org/10.1006/aima.1993.1055>.
- [JSV96] André Joyal, Ross Street, and Dominic Verity. “Traced monoidal categories”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 119.3 (1996), pp. 447–468. ISSN: 0305-0041. DOI: [10.1017/S0305004100074338](https://doi.org/10.1017/S0305004100074338).
- [Kel05] G. M. Kelly. “Basic concepts of enriched category theory”. In: *Reprints in Theory and Applications of Categories* 10 (2005). URL: <http://www.tac.mta.ca/tac/reprints/articles/10/tr10abs.html>.
- [Law04] F William Lawvere. “Functorial Semantics of Algebraic Theories and Some Algebraic Problems in the context of Functorial Semantics of Algebraic Theories”. In: *Reprints in Theory and Applications of Categories* 5 (2004), pp. 1–121.
- [Law73] F William Lawvere. “Metric spaces, generalized logic, and closed categories”. In: *Rendiconti del seminario matematico e fisico di Milano* 43.1 (1973), pp. 135–166.
- [Law86] Bill Lawvere. “State categories and response functors”. 1986.
- [Lei04] Tom Leinster. *Higher operads, higher categories*. London Mathematical Society Lecture Note Series 298. Cambridge University Press, Cambridge, 2004. ISBN: 0-521-53215-9. DOI: [10.1017/CB09780511525896](https://doi.org/10.1017/CB09780511525896).

- [Lei14] Tom Leinster. *Basic category theory*. Vol. 143. Cambridge University Press, 2014.
- [LS88] J. Lambek and P. J. Scott. *Introduction to higher order categorical logic*. Vol. 7. Cambridge Studies in Advanced Mathematics. Reprint of the 1986 original. Cambridge University Press, Cambridge, 1988, pp. x+293. ISBN: 0-521-35653-9.
- [Mac98] Saunders Mac Lane. *Categories for the working mathematician*. 2nd ed. Graduate Texts in Mathematics 5. New York: Springer-Verlag, 1998. ISBN: 0-387-98403-8.
- [May72] J Peter May. *The geometry of iterated loop spaces, volume 271 of Lecture Notes in Mathematics*. 1972.
- [McL90] Colin McLarty. "The uses and abuses of the history of topos theory". In: *The British Journal for the Philosophy of Science* 41.3 (1990), pp. 351–375.
- [McL92] Colin McLarty. *Elementary categories, elementary toposes*. Clarendon Press, 1992.
- [MM92] Saunders MacLane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Springer, 1992. ISBN: 0387977104.
- [nLab] *Symmetric monoidal category*. URL: <https://ncatlab.org/nlab/revision/symmetric+monoidal+category/30> (visited on 02/04/2018).
- [NNH99] Flemming Nielson, Hanne R. Nielson, and Chris Hankin. *Principles of Program Analysis*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999. ISBN: 3540654100.
- [PS95] Frank Piessens and Eric Steegmans. "Categorical data specifications". In: *Theory and Applications of Categories* 1.8 (1995), pp. 156–173.
- [Rie17] Emily Riehl. *Category theory in context*. Courier Dover Publications, 2017.
- [Ros90] Kimmo I Rosenthal. *Quantales and their applications*. Vol. 234. Longman Scientific and Technical, 1990.
- [RS13] Dylan Rupel and David I. Spivak. *The operad of temporal wiring diagrams: formalizing a graphical language for discrete-time processes*. 2013. eprint: [arXiv: 1307.6894](https://arxiv.org/abs/1307.6894).
- [RW92] Robert Rosebrugh and R. J. Wood. "Relational Databases and Indexed Categories". In: *Canadian Mathematical Society Conference Proceedings. International Summer Category Theory Meeting*. (June 23–30, 1991). Ed. by R. A. G. Seely. Vol. 13. American Mathematical Society, 1992, pp. 391–407.
- [S+15] Eswaran Subrahmanian, Christopher Lee, Helen Granger, et al. "Managing and supporting product life cycle through engineering change management for a complex product". In: *Research in Engineering Design* 26.3 (2015), pp. 189–217.

- [Sch+17] Patrick Schultz, David I. Spivak, Christina Vasilakopoulou, and Ryan Wisnesky. “Algebraic Databases”. In: *Theory Appl. Categ.* 32 (2017), Paper No. 16, 547–619.
- [Sel10] Peter Selinger. “A survey of graphical languages for monoidal categories”. In: *New structures for physics*. Springer, 2010, pp. 289–355.
- [Shu08] Michael Shulman. “Framed bicategories and monoidal fibrations”. In: *Theory and Applications of Categories* 20.18 (2008), pp. 650–738. arXiv: [0706.1286](https://arxiv.org/abs/0706.1286) [math.CT].
- [Shu10] Michael Shulman. *Constructing symmetric monoidal bicategories*. 2010. arXiv: [1004.0993](https://arxiv.org/abs/1004.0993) [math.CT].
- [Sob] *Graphical Linear Algebra*. URL: <https://graphicallinearalgebra.net/> (visited on 03/11/2018).
- [Spi+16] David I. Spivak, Magdalen R. C. Dobson, Sapna Kumari, and Lawrence Wu. *Pixel Arrays: A fast and elementary method for solving nonlinear systems*. 2016. eprint: [arXiv:1609.00061](https://arxiv.org/abs/1609.00061).
- [Spi12] David I. Spivak. “Functorial data migration”. In: *Information and Computation* 127 (Aug. 2012), pp. 31–51. ISSN: 0890-5401. DOI: [10.1016/j.ic.2012.05.001](https://doi.org/10.1016/j.ic.2012.05.001). arXiv: [1009.1166v4](https://arxiv.org/abs/1009.1166v4) [cs.DB].
- [Spi13] David I. Spivak. “The operad of wiring diagrams: formalizing a graphical language for databases, recursion, and plug-and-play circuits”. In: *CoRR abs/1305.0297* (2013). URL: <http://arxiv.org/abs/1305.0297>.
- [Spi14] David I Spivak. *Category theory for the sciences*. MIT Press, 2014.
- [SS17] Patrick Schultz and David I. Spivak. *Temporal Type Theory: A topos-theoretic approach to systems and behavior*. 2017. arXiv: [1710.10258](https://arxiv.org/abs/1710.10258) [math.CT].
- [SW15a] Patrick Schultz and Ryan Wisnesky. *Algebraic Data Integration*. 2015. eprint: [arXiv:1503.03571](https://arxiv.org/abs/1503.03571).
- [SW15b] David I. Spivak and Ryan Wisnesky. “Relational Foundations for Functorial Data Migration”. In: *Proceedings of the 15th Symposium on Database Programming Languages*. DBPL 2015. Pittsburgh, PA: ACM, 2015, pp. 21–28. ISBN: 978-1-4503-3902-5. DOI: [10.1145/2815072.2815075](https://doi.org/10.1145/2815072.2815075).
- [TG96] Chris Tuijn and Marc Gyssens. “CGOOD, a categorical graph-oriented object data model”. In: *Theoretical Computer Science* 160.1-2 (1996), pp. 217–239.
- [VSL15] Dmitry Vagner, David I. Spivak, and Eugene Lerman. “Algebras of open dynamical systems on the operad of wiring diagrams”. In: *Theory Appl. Categ.* 30 (2015), Paper No. 51, 1793–1822. ISSN: 1201-561X.
- [Wil07] Jan C Willems. “The behavioral approach to open and interconnected systems”. In: *IEEE Control Systems* 27.6 (2007), pp. 46–99.

- [Wis+15] Ryan Wisnesky, David I. Spivak, Patrick Schultz, and Eswaran Subrahmanian. *Functorial Data Migration: From Theory to Practice*. report G2015-1701. National Institute of Standards and Technology, 2015. arXiv: [1502.05947v2](https://arxiv.org/abs/1502.05947v2).
- [Zan15] Fabio Zanasi. “Interacting Hopf Algebras- the Theory of Linear Systems”. Theses. Ecole normale supérieure de lyon - ENS LYON, Oct. 2015. URL: <https://tel.archives-ouvertes.fr/tel-01218015>.

Index

- Π , 87
- Σ , 87

- ad hoc, 37
- adjoint functor theorem, 58
- adjoint functor theorem, 24
- adjunction, 19, 56, 86
 - examples, 87
 - from closure operator, 26
 - relationship to companions and con-joints, 112
- algebraic theory, 148
- AND, 195, 208
- arrow, 10
- associativity, 67

- base of enrichment, 45
- behavior contract, 216
- behavioral approach, 150
- Beyoncé, 40
- binary relation, 130
- booleans, 5, 11, 41, 56
 - alternative monoidal structure, 41
 - as rig, 137
 - as subobject classifier, 194

- categorification, 111, 114
- category, 67
 - FinSet**, 72
 - Set**, 72
 - as database schema, 74
 - compact closed, 121
 - finitely presented, 69
 - functor category, 80
 - hypergraph, 171
 - monoidal, 117
 - poset reflection, 71
 - presented, 69
 - strict symmetric monoidal, 118
 - symmetric monoidal, 117
- change of base, 52
- chemistry, 29, 37, 56, 185
 - catalysis, 38
- closure
 - compact, 121, 152
 - compact implies monoidal, 122
 - hypergraph categories are compact
 - closed, 173
 - monoidal, 55
- closure operator, 25
- co-design
 - diagram, 100, 106
 - problem, 101
- cocone, 95
- codomain, 67
- coequalizer, 165
- coherence, v, 96, 174
 - as bookkeeping, 117
- colimit, 95, 163, 191
 - and interconnection, 165
 - in **Set**, 164
 - presheaves form colimit completion,

- 198
- collage, 104, 112
- commutative diagram, 79
- commutative square, 69, 76, 79
- companion and conjoint, 111
- cone, 93
- context free grammar, 181
- coproduct, 159
- corelation, 122, 130
- cospan, 165
 - as hypergraph category, 172
 - as theory of Frobenius monoids, 170
 - category of, 166
 - composition, 165
 - decorated, 176
- cost, 56
- currying, 86, 193, 211
- cyber-physical system, 127
- dagger, 53
- data migration, 83
- database, 63
 - as interlocking tables, 63
 - data migration, 66
 - instance, 81
 - instances form a topos, 197
 - query, 88, 94
 - schema, 64, 74, 136
- diagram, 79
 - commutative, 79
- domain, 67
- ends of a spectrum, 71
- enriched category, 45
 - change of base, 52
 - enriching over posets vs categories, 119
 - general definition, 119
 - vs category, 72
- epimorphism, 192
- equivalence of categories, 53, 81
- equivalence relation, 8
 - as corelation, 122
- feasibility relation, 102
- feasibility relation, 101
 - as **Bool**-profunctor, 103
- free
 - category, 67, 73, 133
 - monoid, 134
 - poset, 132
 - prop, 134
 - schema, 64
- Frobenius monoid, 169
- function, 2, 8
 - bijection, 130, 135
 - composite, 9
 - injection, 192
 - injective, 8
 - surjective, 8
- functor, 52, 75
 - operad, 183
 - prop, 130
 - symmetric monoidal, 173
- functorial query language, FQL, 65
- functorial semantics, 144
- Galois connection, 19, 22
- generative effect, 2, 18, 28
- graph, 10, 67
 - as functor, 81
 - homomorphism, 83
 - weighted, 49
- group, 70
- Hasse diagram, 5, 10
 - for profunctors, 113
- Hausdorff distance, 48, 58
- identity
 - function, 9
 - morphism, 67
- IMPLIES, 196, 208
- infix notation, 7
- informatics, 40

- initial object, 158
 - as colimit, 163
- interconnection, 156
 - as variable sharing, 127
 - network-type, 157
 - via Frobenius monoids, 168
- internal language, 189
- interval domain, \mathbb{IR} , 213
- isomorphism, 73
 - of posets, 15
- join, 3, 17, 58
 - as coproduct, 160
- Kan extension, 97
- Lawvere's poset, **Cost**, 43
 - is monoidal closed, 57
- limit, 93, 191
 - formula for finite limits in **Set**, 94
- manufacturing, 29, 38
- matrix, 59
 - computing profunctors using multiplication of, 105
 - feasibility, 104
 - multiplication, 59
 - of distances, 50
 - rig of, 137
- meet, 17, 92
- metric space, 49
- metric space, 47, 53
 - as **Cost**-category, 49
- modal operator, 212
- modes of transport, 51
- monoid object, 148
- monomorphism, 192
- monotone map, 14
 - as **Bool**-functor, 53
- natural transformation, 79
- natural numbers, 11, 32, 42
 - as free category, 68
 - as rig, 137
- naturality condition, 79
- navigator, 58, 108
- NOT, 196, 208
- old friends, 138
- open set, 199
- operad, 180
 - algebra, 183
 - of cospans, 182
 - of sets, 182
- opposite
 - category, 73
 - enriched, 53
 - poset, 13
- OR, 195, 208
- partition, 7, 20
 - as surjection, 9
 - from preorder, 12
- pie
 - lemon meringue, 30, 36
- poker, 32
- port graph, 130, 134
 - as morphisms in free prop, 135
- poset, 10
 - as **Bool**-category, 45, 46
 - as category, 70
 - discrete, 11
 - monoidal, 31
 - of open sets, 200
 - of partitions, 12
 - of subobjects, 210
 - partial order, 10
 - skeletal, 10
 - symmetric monoidal, 31, 118
- power set, 12, 174
- predicate, 193, 209
- prediction, 187
- presentation
 - of category, 69
 - of metric space, 49

- of prop, 136
- presheaf, 197
- primordial ooze, 46, 81, 115, 149
- product, 91
 - as limit, 93
 - category, 124
 - of categories, 92
 - of enriched categories, 53
 - of sets, 91
- poset, 13
- profunctor, 103
 - Cost**, 104
 - as bridges, 103, 112
 - category of, 109
 - composition, 108
 - unit, 109, 112
- program semantics, 25, 28
- prop, 129
 - FinSet**, 129
 - expression, 135
 - of R -relations, 151, 172
 - of matrices, 141, 146
 - posetal, 130
- proposition, 43
- pullback, 94, 191
- pullback along a map, 16, 24, 85
- pushforward
 - partition, 21
- pushout, 161
- quantale, 55, 59, 61, 103, 137
 - commutative, 57
 - of open sets, 201
- quantification, 210
- real numbers, 12, 32, 34, 148
 - as metric space, 49
 - topology on, 199
- reflexivity, 8
- relation, 7
- resource
 - poset, 99
- resource theory, 29
- rig, 137
 - and rings, 137
- safety proof, 216
- sheaf, 201
 - condition, 201
 - of sections of continuous function, 204
 - of sections of a function, 202
 - on \mathbb{R} , 214
 - vector field, 205
- Sierpinski space, 200
- signal flow graph, 127
 - and linear algebra, 152
 - general, 151
 - simplified, 138
- skeleton, 53
- snake equations, 121
- sound and fury, 120
- spider, 169
- subobject classifier, 193
 - in \mathbb{R} , 215
 - in **Set**, 194
 - in sheaf topos, 207
- superdense nugget from outer space, 194
- symmetry, 8
- terminal object, 90
- theory
 - of hypergraph props, 184
 - of monoids, 149
- topological space, 199
- topos, 206
 - of sets, 190
- total order, 11
- transitivity, 8
- tree of life, 14
- triangle inequality, 47
- type theory, 213
- unitality, 67

- universal property, 136
- universal property, 90, 91, 96, 133, 158
- upper set, 13, 16

- vertex, 10

- wiring diagram, 33
 - as graphical proofs, 35
 - for categories, 115
 - for hypergraph categories, 170
 - for monoidal categories, 116
 - for monoidal posets, 34