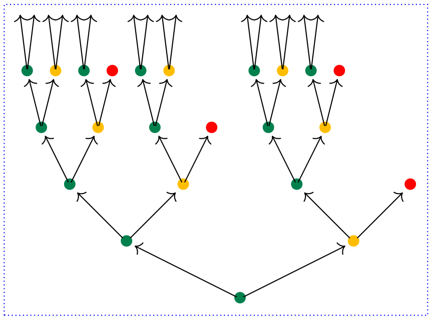
Polynomial Functors: A General Theory of Interaction



David I. Spivak Nelson Niu

(Last updated: September 8, 2021)

This page intentionally left blank.

David I. Spivak
Topos Institute
Berkeley, CA

Nelson Niu
University of Washington
Seattle, WA

To André Joyal

Contents

Preface

> The proposal is also intended to [serve] equally as a foundation for the academic, intellectual, and technological, on the one hand, and for the curious, the moral, the erotic, the political, the artistic, and the sheerly obstreperous, on the other.
>
> –Brian Cantwell Smith
> *On the Origin of Objects*

> For me, though, it is difficult to resist the idea that space-time is not essentially different from matter, which we understand more deeply. If so, it will consist of vast numbers of identical units—"particles of space"—each in contact with a few neighbors, exchanging messages, joining and breaking apart, giving birth and passing away.
>
> –Frank Wilczek
> *Fundamentals*

Why this book

Brief history of book

DJM visited DS in Boston for almost all of 2020. Mutual excitement over different but related things. DS super excited about **Poly**, DJM super excited about doctrines and later paradigms.

DJM encodes something like Poly in Idris and teaches Idris to DS, who having just learned Haskell (thanks Bartosz!) quickly ports the ideas to the strict **Poly** setting. We discussed writing a book together—a T-shape, consisting of a broad generalization with paradigms and a deep dive into **Poly**—after ACT2020, in July.

DS realized comonoids were important to the story. Joachim reminded DS that Richard Garner talked about cofree comonoids at a recent CT. Richard's HottEST talk was completely mind blowing for DS. Worlds collided.

Nelson joins David as undergraduate student researcher and very quickly taught David lots of cool stuff about comonoids and bimodules. Great synergy there.

DS invented poly boxes, and everything started going faster—very fast-paced and relevant research while writing a book is not so good, so it got disjointed. The writing was going quickly, if not particularly coherent, but began to falter in mid-August when DS's hands began to hurt and DJM's teaching duties started. The pace crawled.

DS personally hired Nelson as a typist. Nelson had been so helpful earlier, DS asks DJM if NN can be an author. DJM and DS realize the book would be better as two, DS peels off his part.

DS asked NN if he was willing to join in the effort to make a quality product, and he was. They started in Jan 2021. They decided to teach a course about it at the Topos Institute in Jul 2021.

Acknowledgments

Special thanks to David Jaz Myers, a brilliant colleague, a wonderfully fun conversation partner, a congenial housemate, a superb chef, and an all-around good guy.

Thanks go to John Baez, Spencer Breiner, Kris Brown, Valeria de Paiva, Brendan Fong, Richard Garner, Neil Ghani, Tim Hosgood, Owen Lynch, Joachim Kock, Sophie Libkind, Joshua Meyers, Dominic Orchard, Evan Patterson, Juliet Szatko, Tish Tanski, Todd Trimble, and Christian Williams.

Part I

The category of polynomial functors

Chapter 1

Introduction

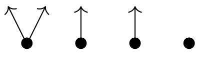
> It is a treasury box!
> Full of unexpected connections!
> It is fascinating!
> I will think about it.
>
> –André Joyal, Summer 2020,
> personal communication.

1.1 Perspectives on polynomials

In this book we will investigate a remarkable category called **Poly**. We will see its intimate relationships with dynamic processes, decision-making, and the storage and transformation of data. But our story begins with something quite humble—high school algebra:

$$y^2 + 2y + 1 \qquad \textit{polynomial} \tag{1.1}$$

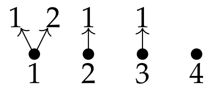
Such polynomials will form the objects of **Poly**. All our polynomials will involve one variable, y, chosen for reasons we'll explain soon. Polynomials in one variable can be drawn as a forest of mini-trees:



forest (1.2)

More technically, each mini-tree—a rooted tree whose *leaves* (the arrows) are all children of the *root* (the solid dot)—is called a *corolla*. A union of trees is called a *forest*, so our polynomials can be viewed as forests of corollas, which we will call *corolla forests*. Each corolla in (1.2) corresponds to a *pure-power summand* of the form y^A in the polynomial given in (1.1): the corolla with 2 leaves corresponds to y^2; the two corollas with 1 leaf each correspond to the two copies of $y = y^1$; and the corolla with no leaves corresponds to $1 = y^0$.

We can label the roots and leaves of our forest, like so:



This suggests yet another depiction of the polynomial (1.1): as a *dependent set*[1] $(p[i])_{i \in I}$. Here $I = \mathsf{4} = \{1, 2, 3, 4\}$[2], the set of roots, and

$$p[1] = \mathsf{2} = \{1, 2\}, \quad p[2] = \mathsf{1} = \{1\}, \quad p[3] = \mathsf{1} = \{1\}, \quad p[4] = \mathsf{0} = \varnothing, \qquad \textit{arena} \tag{1.3}$$

so that for each root $i \in I$, the set $p[i]$ consists of the leaves at that root. We call the entire dependent set $(p[i])_{i \in I}$ an *arena*. Each element $i \in I$ is a *position* in the arena, and each element $d \in p[i]$ is a *direction* at position i. So the relationship between the arena perspective and the forest picture is that positions are roots and directions are leaves.

We will sometimes refer to the index set I as the *position-set* of p, each element $i \in I$ as a *p-position*, each set $p[i]$ as the *direction-set* of p at i, and each element $d \in p[i]$ as a *$p[i]$-direction*.

Throughout this book, we will see several other perspectives from which we can view polynomials. Here is a table of terminology, capturing five different perspectives from which we may view our objects of study. The first row shows the algebraic notation, as in (1.1); the second row shows the dependent set terminology, as in (1.3); the third shows the pictorial terminology of trees, as in (1.2); the fourth shows dynamical systems terminology, which we will explore in Chapter 3; and the fifth row shows decision-making terminology, which we will introduce in Section 1.3.

Polynomial Terminology

algebra	$p \coloneqq \sum_{i \in p(1)} y^{p[i]}$	$i \in p(1)$	$d \in p[i]$
dependent sets	arena	position	direction
tree pictures	corolla forest	root •	leaf ↑
dynamics	(mode-dependent) interface	output	input
decisions	menu	decision	option

(1.4)

Remark 1.5. Though a polynomial will turn out to be a *functor*, while an arena is a *dependent set*, they are so closely related that we often do not make a distinction between a polynomial p and its arena $(p[i])_{i \in I}$; they are essentially two different syntaxes for the same object. For example, we may often directly refer to the positions and directions of a polynomial, when we mean the positions and directions of its associated arena.

On the other hand, while *arenas* and *dependent sets* are exactly the same mathematical object, with the same syntax, we will stick to using the term "arena" when referring to them as objects in our categories of interest that coincide with polynomials, while we will use the term "dependent set" for more general set-theoretic purposes. For

[1] Perhaps better known as an *indexed collection* (or *family*) *of sets*. But we refer to these as *dependent sets* to compare them to the usual sets we are familiar with from set theory, the same way dependent types compare to the usual types we are familiar with from type theory.

[2] In standard font, 4 represents the usual natural number. In sans serif font, $\mathsf{4}$ represents the set $\mathsf{4} = \{1, 2, 3, 4\}$ with 4 elements.

example, we will refer to the positions and directions of arenas, but not the positions and directions of dependent sets; on the other hand, we will refer to the sum or product of a single dependent set when we mean the set-theoretic disjoint union or cartesian product of a collection of sets, but not the sum or product of a single arena in this way.

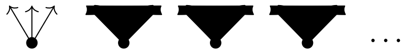
Exercise 1.6 (Solution here). Consider the polynomial $p \coloneqq 2y^3 + 2y + 1$ and the associated corolla forest and arena.

1. Draw the polynomial p as a corolla forest.
2. How many roots does this forest have?
3. How many positions in the arena does this represent?
4. For each corolla in the forest, say how many leaves it has.
5. For each position in the arena, how many directions does it have? ◊

Exercise 1.7 (Solution here). Consider the polynomial $q \coloneqq y^8 + 4y$.

1. Does the polynomial q have a pure-power summand y^2?
2. Does the polynomial q have a pure-power summand y?
3. Does the polynomial q have a pure-power summand $4y$? ◊

One feature that sets our polynomials apart from the polynomials we are familiar with from high school algebra is that the coefficients and exponents are not, strictly speaking, numbers; rather, they are sets, like $1 = \{1\}$ and $2 = \{1, 2\}$. In fact, they can be arbitrary sets, as in $By^A + Dy^C$ for sets A, B, C, D, including infinite ones, as in $\mathbb{R}y^{\mathbb{N}} + 2^{\mathbb{R}}y^{\mathbb{R}}$, leading to an infinite number of roots or leaves per root. Any finite or infinite sum of pure-power summands, each with a finite or infinite set as an exponent, is still a polynomial. Of course, this makes their forests rather unwieldy to draw, but they can be approximated. We sketch the polynomial $y^3 + \mathbb{N}y^{[0,1]}$ as a forest below.



Exercise 1.8 (Solution here). If you were a suitor choosing the corolla forest you love, aesthetically speaking, which would strike your interest? Answer by circling the associated polynomial:

1. $y^2 + y + 1$
2. $y^2 + 3y^2 + 3y + 1$
3. y^2
4. $y + 1$
5. $(\mathbb{N}y)^{\mathbb{N}}$
6. Sy^S
7. $y^{100} + y^2 + 3y$
8. $y + 2y^4 + 3y^9 + 4y^{16} + \cdots$

9. Your polynomial's name p here.

Any reason for your choice? Draw a sketch of your forest. ◊

Before we can really get into this story, let's summarize where we're going: polynomials are going to have really surprising applications to dynamics, decisions, and data. We speak superlatively of **Poly**:

The category of polynomials is a jackpot. Its beauty flows without bound

but we have not yet begun to deliver. So let's introduce some of the applications and mathematics to come.

1.2 Dynamical systems

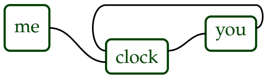
When we say "dynamical system," we are referring to a concept that may be familiar: a machine that stores an internal state. The machine may return output according to its current state, while it may also receive input that updates this state.

For example, the internal state of a digital clock may consist of the time and the display format ("12-hour" vs. "24-hour"). If the time is six minutes past noon and the display format is "12-hour," the clock will display "12:06pm" as its output. If the time is twenty minutes till midnight and the display format is "24-hour," the clock will instead display "23:40."

Meanwhile, the clock may receive input via one button that increments its time by one minute and another button that toggles between the two display formats. At the press of a button, the internal state will change depending on which button was pressed and what the previous internal state was.

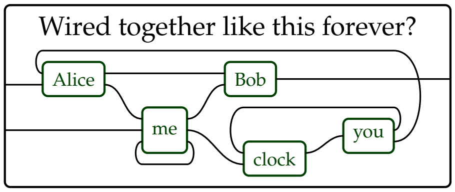
So our digital clock is a very simple dynamical system. We can think of its input buttons and its output display as the way in which the clock interacts with the outside world—its *interface*.

Here's one way the clock might interact with the outside world: my finger is on the minute button, while your finger is on the format button, and the clock is facing you so that only you can read its display. In fact, we could think of ourselves as a couple of (particularly complex) dynamical systems as well, with each of our interfaces connected with the interface of the clock. We could model this scenario with the following picture, called a *wiring diagram*, showing how each system can receive input from and send output to other systems in a particular interaction pattern:



Of course, there is a lot going on in the world around us that we haven't drawn. We each have some input ports: our eyes, our ears, etc., and some output ports: our speech, our gestures, etc. We can connect with other systems: our family, our colleagues, etc.

And we can think of all of these systems as subsystems of one larger system interacting with the world.



(1.9)

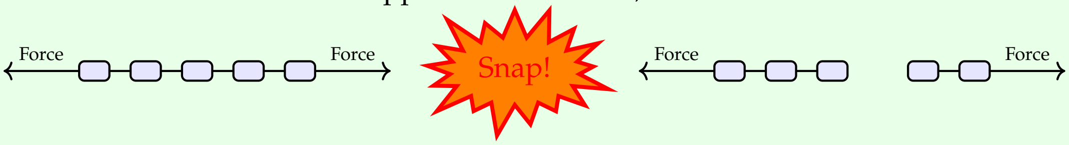
We wrote a little question for you at the top of the diagram. Isn't there something a little funny about the way we've connected these systems? Maybe for very simple machines, you would wire things together once and they'd stay like that for the life of the machine. But you could turn your eyes away from the clock to look at Bob, Alice drops her connection to me for weeks at a time, and I would really like to be able to lift my finger off of the clock to do something else. So

the way systems connect can change over time.

In fact, **Poly** will let us express this.

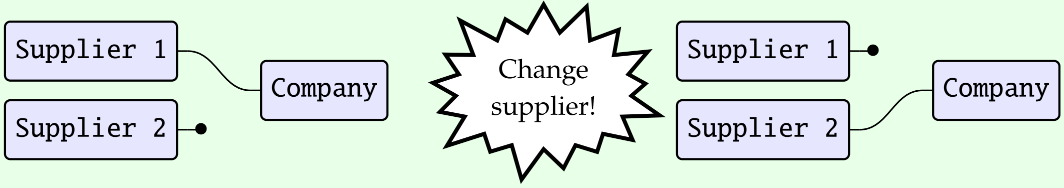
Example 1.10. Here are some familiar circumstances where we see interaction patterns changing over time.

1. When too much force is applied to a material, bonds can break:



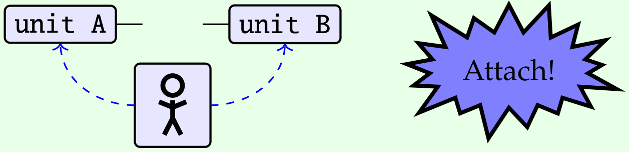
In materials science, the Young's modulus accounts for how much force can be transferred across a material as its endpoints are pulled apart. When the material breaks, the two sides can no longer feel evidence of each other. Thinking of pulling as sending a signal (a signal of force), we might say that the ability of internal entities to send signals to each other—the connectivity of the wiring diagram—is being measured by the Young's modulus. It will also be visible within **Poly**.

2. A company may change its supplier at any time:



The company can get widgets either from supplier 1 or supplier 2; we could imagine this choice is completely up to the company. The company can decide based on the quality of widgets it has received in the past: when the company gets a bad widget, it updates an internal variable, and sometimes that variable passes a threshold making the company switch states. Whatever its strategy for deciding, we should be able to encode it in **Poly**.

3. When someone assembles a machine, their own outputs dictate the connection pattern of the machine's components.



Have you ever assembled something? Your internal states dictate the interaction pattern of some other things. We can say this in **Poly**.

All of the examples discussed here will be presented in some detail once we have the requisite mathematical theory (Examples 3.72, 3.74, and 3.75).

Exercise 1.11 (Solution here). Think of another example where systems are sending each other information, but where who the information is being sent to or received from can change based on the states of the systems involved. You might have more than two, say $\mathbb{R}$-many, different interaction patterns in your setting. $\Diamond$

But there's more that's intuitively wrong or limiting about the picture in (1.9). Ever notice how you can change how you interface with the world? Sometimes I close my eyes, which makes that particular way of sending me information inaccessible: that port vanishes, and you need to use your words. Sometimes I'm in a tunnel and my car can't receive a radio signal. Sometimes I extend my hand to give or receive an object from another person, but sometimes I don't. So

a system's interface itself can change over time.

We will be able to say all this using **Poly** as well.

And there's even more that's wrong with the above description. Namely, when I use my muscles or mouth to express things, my very position changes: my tongue moves, my body moves. The display of an analog clock is literally the positions of its hands. So

the output of a system is essentially the position it takes.

Moreover, when I move my eyes, that's something you can actually see—you can tell if I'm looking at you. When I turn around, I see different things, and *you can notice I'm turned around*! In other words,

the range of inputs a system can receive depends on the position it currently outputs.

This is integral to our model of dynamical systems in **Poly**, and why we say that outputs correspond to positions and inputs to directions—with an entire interface represented by a polynomial.

Example 1.12. Imagine a million little eyeballs, each of which has a tiny brain inside it, all together in a pond of eyeballs. All that an individual eyeball e can do is open and close. When e is open, it can make some distinction about all the rest of the eyeballs in view: maybe it can count how many are open, or maybe it can see whether just one certain eyeball e' is open or closed. But when e is closed, it can't see anything; whatever else is happening, it's all the same to e. All it can do in that state is process previous information.

Each eyeball in this system will correspond to the polynomial $y^{\mathsf{n}} + y$, which consists of two positions: an "open" position with n-many possible inputs it may perceive, and a "closed" position with only one. For simplicity, we could assume $n = 2$, so that each eyeball makes a single yes-no distinction whenever it's open.

The point, however, is that any other eyeball may be capable of noticing if e is open or closed. We can imagine some interesting dynamics in this system, e.g. waves of openings or closings sweeping over the group, a ripple of openings expanding through the pond.

Talk about real-world applications!

Exercise 1.13 (Solution here). Give another example of a system where the range of possible inputs the system can receive is dependent on what output the system is currently providing. ◊

Hopefully you now have an idea of what we mean by *mode-dependence*: interfaces and interaction patterns changing over time, based on the states of all the systems involved. We'll see that **Poly** speaks about mode-dependent systems and interaction patterns in this sense.

Remark 1.14. We ended Example 1.12 by joking about "real-world applications," because a pond of eyeballs is about the most bizarre thing one can imagine. But recall Nobel physicist Frank Wilczek's quote from the preface:

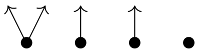
> For me, though, it is difficult to resist the idea that space-time is not essentially different from matter, which we understand more deeply. If so, it will consist of vast numbers of identical units—"particles of space"—each in contact with a few neighbors, exchanging messages, joining and breaking apart, giving birth and passing away.

Suppose the world was made out of a vast number of identical units, each with its own behavior, able to connect and disconnect with neighbors, and even disappear from the world of cause and effect. We may not even be interested in what our world is actually made of—just what these units are able to do. Is there such an elementary unit that could produce all other dynamical systems? The $y^2 + y$ eyeballs give a sense of a very simple interface—open and perceiving a single distinction about the world, or closed and making no distinctions—that we could imagine building an entire world from.

It turns out **Poly** is very versatile in its applications. In the next section, we'll show how it relates to decision-making.

1.3 Decisions

We return to the example polynomial $y^2 + 2y + 1$ from (1.1) and its corresponding corolla forest, in which positions are expressed as roots and directions are represented as leaves:

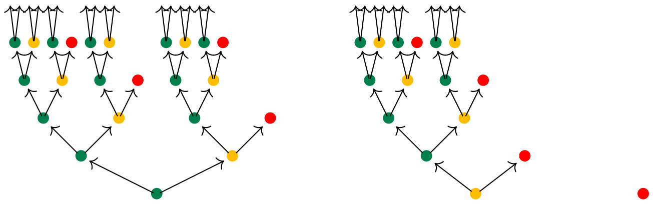


(1.15)

Concretely, we might think of each position as representing a *decision*. Associated to every decision is a set of *options* (directions). The four decisions we exhibit in (1.15) are particularly interesting: they respectively have two options, one option, one option, and no options. Having two options is familiar from life—it's the classic yes/no decision—as well as from Claude Shannon's Information Theory. Having one option is also familiar theoretically and in life: "sorry, ya just gotta go through it." Having no options is when you actually don't get through it: an impossible decision, a sort of "dead end."

This perspective will prove insightful in Section 2.3, when we start to discuss the *morphisms* between two polynomials. As a sneak peek, it turns out that a morphism is just a way of *delegating* decisions from one menu to another. We'll cover this in more detail in Section 2.3.

Now consider the following three trees. The first two are infinite, but that's hard to draw, so we've included just the first five levels:

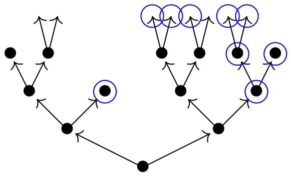


These are patterned examples—and we'll understand what this pattern is more clearly in **??**—of what we will call *decision streams*. Decision streams form the objects of a category with very nice properties (it's a topos), which we call $\mathbf{Sys}(y^2 + 1)$. The idea is that these trees are built up of smaller corollas, each of which has either two options, corresponding to y^2, or no options, corresponding to $1 \cong y^0$. We say that each such decision stream has type $y^2 + y^0$.

Exercise 1.16 (Solution here).

1. Draw the first three levels of a decision stream of type $y^2 + y^0$.
2. Draw the first four levels of a decision stream of type y.
3. Draw a the first three levels of a decision stream of type $\mathbb{N}y^2$ by labeling every node with a natural number. ◊

But decisions aren't just about choosing; they're also about trying to accomplish something. The logic of accomplishment is exceptionally rich in this setting. We will concentrate on what we call a *win condition*, which is an induced subgraph of the decision stream with the property that if n is a winning node, then any child of n is also a winning node.



More formally, these are called *sieves*. They form the elements of a logical system called a Heyting algebra: you can take any two sieves and form the intersection or union (which correspond to AND and OR), or even things like implication, negation, and existential and universal quantification. This will give us a calculus of win-conditions for any type p decision stream.

1.4 Data

Data is information, maybe thought of as quantized into atomic pieces, but where these atomic pieces are somehow linked together according to a conceptual structure. When a person or organization uses certain data repeatedly, they often find it useful to put their data in a database. This requires organizing the little pieces into a conceptual structure. So when you hear "database," just think of it as a conceptual structure filled with examples.

To fix a mental image, let's say that you need to constantly look up employees, what department they're in, who the admin person is for that department, who their manager is, etc. Here's an associated database

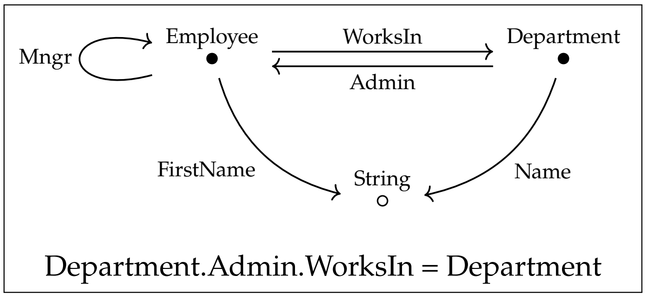
Employee	**FirstName**	**WorksIn**	**Mngr**
1	Alan	101	2
2	Ruth	101	2
3	Carla	102	3

Department	**Name**	**Admin**
101	Sales	1
102	IT	3

(1.17)

We can see it as being associated to the following conceptual scheme, also called a *schema*:

$\mathcal{C} :=$



(1.18)

The equation at the bottom says that for any department d, if you ask for the admin person and see which department they work in, it's required to be d.

What's called $\mathcal{C}$ in (1.18) is a *finitely presented category*. The objects of the category are the points, while each arrow corresponds to a morphism of $\mathcal{C}$ (there are additional morphisms, including identity morphisms, that are not depicted). The equation at the bottom indicates that composing the morphism Admin with the morphism WorksIn yields the identity morphism on the object Department.

The database instance presented in (1.17) then corresponds to a functor $I\colon \mathcal{C} \to \mathbf{Set}$. For example, I sends the object Employee $\in \mathcal{C}$ to the set $I(\text{Employee}) := \{1, 2, 3\}$, the entries in the Employee column of the left table.

The functor also sends the morphism Mngr: Employee $\to$ Employee to the function $I(\text{Mngr})\colon \{1, 2, 3\} \to \{1, 2, 3\}$ that sends each entry in the Employee column to its corresponding entry in the Mngr column. So $I(\text{Mngr})(1) = 2$, $I(\text{Mngr})(2) = 2$, and $I(\text{Mngr})(3) = 3$.

A functor $\mathcal{C} \to \mathbf{Set}$ is called a *copresheaf on* $\mathcal{C}$. So the story of database schemas and their data can be based on the story of categories and their copresheaves.

Exercise 1.19 (Solution here). As above, we define the finitely presented category $\mathcal{C}$ according to (1.18) and the copresheaf I on $\mathcal{C}$ according to (1.17).

1. What is $I(\text{Department})$?
2. What is $I(\text{Admin})$?
3. Composing Admin with FirstName yields a morphism from Department to String that we denote by Admin.FirstName. What is $I(\text{Admin.FirstName})$?
4. Say we require that managers work in the same department as the employees they oversee. Write down an equation in $\mathcal{C}$ (like the one at the bottom of (1.18)) that expresses this condition.
5. How might we define $I(\text{String})$? ◊

There's a very important thing that we do with databases: we query them. We ask them questions like "tell me the First Name of every Employee that's either the Admin of the Sales department or their Manager."

```
FOR     d: Department, e: Employee
WHERE   Name(d)="Sales" AND
        (e=Admin(d) OR e=Mngr(Admin(d)))
RETURN  FirstName(e)
```

This sort of question is formally called a "union of conjunctive queries." We will see this sort of query is intimately connected with **Poly**. We will also see how databases can be conceived in terms of dynamical systems.

1.5 Implementation

Everything we talk about can actually be implemented in a computer without much difficulty, at least if you have access to a language that supports dependent types, such as Agda or Idris.

What we have been calling polynomials—things like $y^2 + 2y + 1$—are often called *containers* in the computer science literature. A container consists of a type S, usually called the type of *shapes*, and a type $P(s)$ for each term $s : S$, called the type of *positions* in shape s. It's mildly unfortunate that the names clash with our own: for us a container-shape is a position and a container-position is a direction.

Luckily, the Agda code is pretty easy to understand.

```
record Arena : Set where  -- an arena consists of
   field                      -- two fields
     pos : Set                -- one called pos, a set; and
     dir : pos -> Set         -- one called dir,
                              -- a set for each element of the set pos
```

1.6 Mathematical theory

The applications of **Poly** are quite diverse and interesting, encapsulating dynamics, data, and decisions. However it is how the mathematics of **Poly** supports these applications that is so fantastic. For experts, here are some reasons for the excitement.

Proposition 1.20. **Poly** has all products and coproducts and is completely distributive. **Poly** also has exponential objects, making it a bicartesian closed category. It therefore supports the simply typed lambda calculus.

Proof. We will prove that **Poly** has coproducts in Proposition 2.50, that it has products in Proposition 2.79, that it is completely distributive in Corollary 2.81, and that it has exponential objects in Theorem 4.30. □

Proposition 1.21. Beyond the cocartesian and cartesian monoidal structures $(0,+)$ and $(1,\times)$, the category **Poly** has two additional monoidal structures, denoted $(y,\otimes)$ and $(y,\circ)$, which are together duoidal.[a] Moreover $\otimes$ is a closed monoidal structure that distributes over coproducts, while $\circ$ is a left coclosed monoidal structure that preserves connected limits.

[a] We will follow the convention of writing the tensor unit before the tensor product when specifying a monoidal structure.

Proof. We will define $\otimes$ in Definition 2.88 and prove that $(y,\otimes)$ is a monoidal structure on **Poly** in Proposition 2.96, and we will define $\circ$ in Definition 5.1 and prove that $(y,\circ)$ is a monoidal structure on **Poly** in Corollary 5.5. Then we will show that $\circ$ is duoidal over $\otimes$ in **??**.

In Proposition 2.101, we will show that $\otimes$ distributes over coproducts. Then in Proposition 3.87, we will prove that $\otimes$ is closed. In Proposition 5.46, we will show that $\circ$ is left coclosed, and in **??**, we will show that $\circ$ preserves connected limits. □

Proposition 1.22. **Poly** has an adjoint quadruple with **Set** and an adjoint pair with $\mathbf{Set}^{\mathrm{op}}$:

$$\begin{tikzcd}[column sep=large]
\mathbf{Set} \arrow[r, shift left=1, "A" description] \arrow[r, shift right=5, "Ay"'] & \mathbf{Poly} \arrow[l, shift right=5, "p(0)"'] \arrow[l, shift left=1, "p(1)" description]
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\mathbf{Set}^{\mathrm{op}} \arrow[r, shift left=2, "y^A"] \arrow[r, phantom, "\Leftarrow"] & \mathbf{Poly} \arrow[l, shift left=2, "\Gamma(p)"]
\end{tikzcd}\ .$$[a]

Each functor is labeled by where it sends $p \in$ **Poly** or $A \in$ **Set**; in particular, $\Gamma(p) := \mathbf{Poly}(p,y)$.

[a] We use the notation $\begin{tikzcd} \mathcal{C} \arrow[r, shift left=2, "L"] \arrow[r, phantom, "\Rightarrow"] & \mathcal{D} \arrow[l, shift left=2, "R"] \end{tikzcd}$ to denote an adjunction $L \dashv R$. The double arrow, always pointing in the same direction as the left adjoint, indicates both the unit $\mathcal{C} \Rightarrow R \circ L$ and the counit $L \circ R \Rightarrow \mathcal{D}$ of the adjunction.

Proof. We will prove that **Poly** has an adjoint quadruple with **Set** in Theorem 4.4, and that it has an adjoint pair with $\mathbf{Set}^{\mathrm{op}}$ in Proposition 4.12. □

There's a lot we're leaving out of this summary, just so we can hit the highlights.

Proposition 1.23. The functor $\mathbf{Poly} \to \mathbf{Set}$ given by $p \mapsto p(1)$ is a monoidal fibration.

In fact it's a monoidal $*$-bifibration in the sense of [Shu08]. But here's where things get really interesting.

Proposition 1.24 (Ahman-Uustalu)**.** Comonoids in $(\mathbf{Poly}, y, \circ)$ are categories (up to isomorphism).

Proposition 1.25. The category $\mathbf{Comon}(\mathbf{Poly})$ has finite coproducts and products, and coproducts in $\mathbf{Comon}(\mathbf{Poly})$ agree with those in $\mathbf{Cat}$.

Proposition 1.26. The forgetful functor $U\colon \mathbf{Comon}(\mathbf{Poly}) \to \mathbf{Poly}$ has a right adjoint

$$\begin{tikzcd} \mathbf{Poly} \arrow[r, shift left=1.5ex, "\mathscr{T}_{-}"] \arrow[r, phantom, "\Leftarrow"] & \mathbf{Comon}(\mathbf{Poly}) \arrow[l, shift left=1.5ex, "U"] \end{tikzcd}$$

called the *cofree comonoid* construction. It is lax monoidal with respect to $\otimes$.

Proposition 1.27. The category $\mathbf{Comon}(\mathbf{Poly})$ has a third symmetric monoidal structure $(y, \otimes)$, and the forgetful functor $U\colon (\mathbf{Comon}(\mathbf{Poly}), y, \otimes) \to (\mathbf{Poly}, y, \otimes)$ is strong monoidal.

Proposition 1.28. For any polynomial p, the category

$$\mathbf{Sys}(p) \cong [\mathscr{T}_p, \mathbf{Set}]$$

of dynamical systems on p forms a topos.[a]

[a] We use the notation $[\mathcal{C}, \mathcal{D}]$ to denote the category of functors $\mathcal{C} \to \mathcal{D}$.

The proposition above indicates that there is a full-fledged logic of dynamical systems inhabiting any interface, while the following proposition implies that these logics can be combined and compared.

Proposition 1.29. A morphism $p \to q$ of polynomials induces a pre-geometric morphism between their respective toposes:

$$\begin{tikzcd} \mathbf{Sys}(p) \arrow[r, shift left=1.5ex] \arrow[r, phantom, "\Rightarrow"] & \mathbf{Sys}(q) \arrow[l, shift left=1.5ex] \end{tikzcd} .$$

The following propositions suggest that the whole story of dynamics carries a strong connection to database theory.

Proposition 1.30. Suppose that the category $\mathcal{C}$ corresponds under Proposition 1.24 to the comonoid $\mathscr{C}$. Then there is an equivalence of categories

$$\mathbf{Bimod}(\mathscr{C}, 0) \cong [\mathcal{C}, \mathbf{Set}]$$

between $(\mathscr{C}, 0)$-bimodules[a] and $\mathcal{C}$-copresheaves.

[a] Technically, what we are calling bimodules over a pair of comonoids are really bi*co*modules, but we will use the simpler term throughout.

We will use the convention that the comonoid $\mathscr{C}$ corresponds to category $\mathcal{C}$ under Proposition 1.24, and similarly for $\mathscr{D}$ and $\mathcal{D}$, etc.

Proposition 1.31. For any category $\mathcal{C}$, the category of left $\mathscr{C}$-modules is equivalent to the category of functors $\mathcal{C} \to \mathbf{Poly}$:

$$\mathbf{Bimod}(\mathscr{C}, y) \cong \mathbf{Fun}(\mathcal{C}, \mathbf{Poly}).$$

Proposition 1.32 (Garner)**.** For any categories $\mathcal{C}$ and $\mathcal{D}$, there is an equivalence of categories

$$\mathbf{Bimod}(\mathscr{C}, \mathscr{D}) \cong \mathbf{pra}([\mathcal{D}, \mathbf{Set}], [\mathcal{C}, \mathbf{Set}])$$

between that of bimodules between comonoids in **Poly** and parametric right adjoints between copresheaf categories.

If you skipped over any of that—or all of that—it'll be no problem whatsoever! We will cover each of the above results in detail over the course of this book. There are many avenues for study, but we need to push forward.

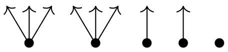
We'll begin in the next chapter.

1.7 Exercise solutions

Solution to Exercise 1.6.

We consider the polynomial $p := 2y^3 + 2y + 1$.

1. Here is p drawn as a forest of corollas (note that the order in which the corollas are drawn does not matter):



2. It has five (5) roots.
3. It represents five positions, one per root.
4. The first and second corollas have three leaves, the third and fourth corollas have one leaf, and the fifth corolla has no leaves.
5. The direction-set at each position is the same as the set of leaves of each corolla, so just copy the answer from #4, replacing "corolla" with "position" and "leaf" with "direction." Sheesh! Who wrote this.

Solution to Exercise 1.7.

We refer to the polynomial $q := y^8 + 4y$.

1. No, q does not have y^2 as a pure-power summand.
2. Yes, q does have y as a pure-power summand.
3. No, q does not have $4y$ as a pure-power summand, because $4y$ is not a pure-power! But to make amends, we could say that $4y$ is a summand; this means that there is some q' such that $q = q' + 4y$. So $3y$ is also a summand, but $5y$ and y^2 are not.

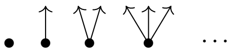
Solution to Exercise 1.8.

Aesthetically speaking, here's the associated polynomial of a beautiful corolla forest:

$$y^0 + y^1 + y^2 + y^3 + \cdots$$

It's reminiscent (and formally related) to the notion of lists: if A is any set, then $A^0 + A^1 + A^2 + \cdots$ is the set $\mathsf{List}(A)$ of lists (i.e. finite ordered sequences) with entries in A.

Here's a picture of this lovely forest:



Solution to Exercise 1.11.

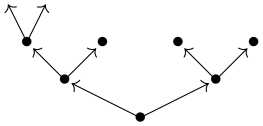
When I am carrying my phone in my house, my phone will connect to my Wi-Fi router. But my phone may send me a reminder that tells me to leave my house. Once I carry my phone far enough away, it will disconnect from my router and connect to a cellular network instead. While I am outside, I might then press a button on my phone to disconnect it from the cellular network to reduce my data usage, so that it is no longer connectd to any network.

Solution to Exercise 1.13.

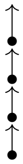
Consider a computer application whose output is an image displayed on the screen. This image consists of buttons I can click, and each button-click is a possible input that the application can receive. Clicking a button will alter the display, including which buttons are now available to be clicked. For example, when I click the "File" button, the "File" menu will appear, allowing me to click other buttons like the "New Window" button. So the set of inputs I can send to the system (i.e. buttons I can click) is directly dependent on the output the system sends to me (i.e. the image displayed on the screen).

Solution to Exercise 1.16.

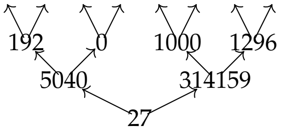
1. Here are the first three levels of a decision stream of type $y^2 + y^0$.



2. Here are the first four levels of a decision stream of type y.



3. Here are the first three levels of a decision stream of type $\mathbb{N}y^2$ where we indicate the position of each node by labeling it with a natural number.



Solution to Exercise 1.19.

We refer to (1.18) and (1.17) to characterize the category $\mathcal{C}$ and the functor $I : \mathcal{C} \to \mathbf{Set}$.

1. Here Department is an object of $\mathcal{C}$, so $I(\text{Department})$ is a set. From the Department column in the table on the right of (1.17), we observe that $I(\text{Department}) = \{101, 102\}$.
2. Here Admin is an morphism of $\mathcal{C}$ from Department to Employee, so $I(\text{Admin})$ is a function from $I(\text{Department}) = \{101, 102\}$ to $I(\text{Employee}) = \{1, 2, 3\}$. From the Admin column in the table on the right of (1.17), we observe that $I(\text{Admin})(101) = 1$ and $I(\text{Admin})(102) = 3$.

3. By functoriality, I(Admin.FirstName) is the function I(Admin) composed with I(FirstName). We have that I(Admin) sends 101 to 1 and 102 to 3, while the FirstName column tells us that I(FirstName) sends 1 to Alan and 3 to Carla. So I(Admin.FirstName) sends 101 to Alan and 102 to Carla.
4. If every employee works in the same department as their manager, then Employee.WorksIn = Employee.Mngr.WorksIn.
5. The name suggests that I(String) is the set of all possible strings of characters. Perhaps this could be defined as $\bigcup_{n \in \mathbb{N}} A^n$, where A is our alphabet of allowed characters, which may include the English letters, spaces, digits, and whatever other characters we allow. At the very least, since FirstName and Name are both morphisms to String, every entry in the FirstName and Name columns must be in I(String). So all we know for sure is that $\{$Alan, Ruth, Carla, Sales, IT$\} \subseteq I$(String).

Chapter 2

Polynomial functors and natural transformations

In this chapter, we will set down the basic category-theoretic story of **Poly**, so that we can have a firm foundation from which to speak about dynamical systems, decisions, and data. We will formally introduce the objects of **Poly**: polynomial functors. We will also study the natural transformations between these functors, which are **Poly**'s morphisms. Finally, we'll present some of **Poly**'s most versatile categorical properties.

But we'll begin by examining a specific kind of polynomial functor that you may already be familiar with—representable functors on the category **Set** of sets and functions. We'll highlight how these functors are closely related to what is arguably the fundamental theorem of category theory, the Yoneda lemma.

2.1 Representable functors and the Yoneda lemma

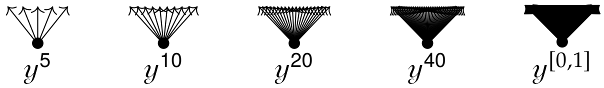
Representable functors form the basis of the category **Poly**. Much of the following theory is applicable for general functors into **Set**; but for now, we are only interested in functors **Set** → **Set**.

Definition 2.1. Given any set S, we denote by $y^S\colon \mathbf{Set} \to \mathbf{Set}$ the functor that sends any set X to the set $X^S = \mathbf{Set}(S, X)$, and sends any function $h\colon X \to Y$ to the function $h^S\colon X^S \to Y^S$, the one that sends $g\colon S \to X$ to $g \mathbin{⨟} h\colon S \to Y$.[a]

We refer to functors of this form as *representable functors*, or simply as *representables*. In particular, we call y^S the functor *represented by S*, and we call S the *representing set* of y^S.

[a] Throughout this text, in any category, given objects A, B, C and morphisms $f\colon A \to B$ and $g\colon B \to C$, we will denote their composite morphism $A \to C$ interchangeably as $f \mathbin{⨟} g$ or $g \circ f$, depending on which is more natural to think about.

The symbol y stands for Yoneda, for reasons we will get to in Lemma 2.10. For now, here are some pictures for your eyes to gaze at; they are the polynomials corresponding to various representables, namely the pure powers:



(2.2)

Example 2.3. The functor that sends every set X to $X \times X$, and sends $h\colon X \to Y$ to $(h \times h)\colon (X \times X) \to (Y \times Y)$, is representable. After all, $X \times X \cong X^2$, so this functor is just a pure power we are already familiar with: y^2.

Exercise 2.4 (Solution here). For each of the following functors **Set** → **Set**, say if it is representable or not; if it is, say what the representing set is.

1. The identity functor $X \mapsto X$, which sends each function to itself.
2. The constant functor $X \mapsto 2$, which sends every function to the identity on 2.
3. The constant functor $X \mapsto 1$, which sends every function to the identity on 1.
4. The constant functor $X \mapsto 0$, which sends every function to the identity on 0.
5. A functor $X \mapsto X^{\mathbb{N}}$. If it were representable, where would it send each function?
6. A functor $X \mapsto 2^X$. If it were representable, where would it send each function?

◊

Proposition 2.5. For any function $f\colon R \to S$, there is an induced natural transformation $y^f\colon y^S \to y^R$; on any set X the X-component $X^f\colon X^S \to X^R$ is given by sending $g\colon S \to X$ to $f \mathbin{⨟} g\colon R \to X$.

Exercise 2.6 (Solution here). Prove that for any function $f\colon R \to S$, what we said was a natural transformation in Proposition 2.5 really is natural. That is, for any function $h\colon X \to Y$, show that the following diagram commutes:

$$\begin{tikzcd}
X^S \arrow[r, "h^S"] \arrow[d, "X^f"'] \arrow[dr, phantom, "?"] & Y^S \arrow[d, "Y^f"] \\
X^R \arrow[r, "h^R"'] & Y^R
\end{tikzcd}$$

◊

Exercise 2.7 (Solution here). Let X be an arbitrary set. For each of the following sets R, S and functions $f\colon R \to S$, describe the X-component of, i.e. the function $X^S \to X^R$ coming from, the natural transformation $y^f\colon y^S \to y^R$.

1. $R = 5$, $S = 5$, $f = \text{id}$. (Here you're supposed to give a function called $X^{\text{id}_5}\colon X^5 \to X^5$.)
2. $R = 2$, $S = 1$, f is the unique function.
3. $R = 1$, $S = 2$, $f(1) = 1$.

4. $R = \mathsf{1}$, $S = \mathsf{2}$, $f(1) = 2$.
5. $R = \mathsf{0}$, $S = \mathsf{5}$, f is the unique function.
6. $R = \mathbb{N}$, $S = \mathbb{N}$, $f(n) = n + 1$. ◊

Exercise 2.8 (Solution here). Show that the construction in Proposition 2.5 is functorial

$$y^{-} \colon \mathbf{Set}^{\mathrm{op}} \to [\mathbf{Set}, \mathbf{Set}], \tag{2.9}$$

as follows.

1. Show that for any set S, we have that $y^{\mathrm{id}_S} \colon y^S \to y^S$ is the identity.
2. Show that for any functions $f \colon R \to S$ and $g \colon S \to T$, we have $y^g \mathbin{⨟} y^f = y^{f ⨟ g}$.

◊

Lemma 2.10 (Yoneda lemma)**.** Given a functor $F \colon \mathbf{Set} \to \mathbf{Set}$ and a set S, there is an isomorphism

$$F(S) \cong \mathsf{Nat}(y^S, F) \tag{2.11}$$

where Nat denotes the set of natural transformations. Moreover, (2.11) is natural in both S and F.

Sketch of proof. For any natural transformation $m \colon y^S \to F$, consider the component $m_S \colon S^S \to F(S)$. Applying it to the identity on S as an element of S^S, we get an element $m_S(\mathrm{id}_S) \in F(S)$.

Conversely, for any element $a \in F(S)$, there is a natural transformation $m^a \colon y^S \to F$ whose X-component is the function $X^S \to F(X)$ given by sending $g \colon S \to X$ to $F(g)(a)$. In Exercise 2.12 we ask you to show that this is indeed natural in X, and that these two constructions, $m \mapsto m_S$ and $a \mapsto m^a$, are mutually inverse. □

Exercise 2.12 (Solution here). Whoever solves this exercise can say they've proved the Yoneda lemma.

1. Show that for any $a \in F(S)$, the maps $X^S \to F(X)$ given as in the proof sketch of Lemma 2.10 are natural in X.
2. Show that the two mappings from the proof sketch of Lemma 2.10 are mutually inverse.
3. Show that (2.11) is natural in F.
4. Show that (2.11) is natural in S.
5. As a corollary of Lemma 2.10, show that the functor $y^{-} \colon \mathbf{Set}^{\mathrm{op}} \to [\mathbf{Set}, \mathbf{Set}]$ from (2.9) is fully faithful—in particular, that there is an isomorphism $S^T \cong \mathsf{Nat}(y^S, y^T)$. For this reason, we call this functor the *Yoneda embedding*. ◊

2.2 Polynomials: sums of representables

We've seen that for any set A, the symbol y^A denotes a functor **Set** → **Set**. We will generalize this by adding representable functors together to form polynomials. In some sense, the name "polynomial" doesn't quite fit: in algebra, polynomials are generally taken to be finite sums, whereas we will use sums that may be infinite. However, we are not the first by far to use the term "polynomial" in this way.

So here's the deal. All of our polynomials will be polynomials in one variable, namely y. Every other letter or number that shows up will represent a set.[1] For example, in the following bizarre polynomial

$$p \coloneqq \mathbb{R}y^{\mathbb{Z}} + \mathsf{3}y^3 + Ay + \sum_{i \in I} Q_i y^{R_i + Q_i^2}, \tag{2.13}$$

$\mathbb{R}$ denotes the set of real numbers, $\mathbb{Z}$ denotes the set of integers, $\mathsf{3}$ denotes the set $\{1, 2, 3\}$, and A, I, Q_i, and R_i denote some arbitrary sets that should have already been defined in order for (2.13) to make sense.

The polynomials we already understand at this point are the pure power polynomials y^A for some set A: they are representable functors $y^A \colon \mathbf{Set} \to \mathbf{Set}$. So to understand general polynomials like p, we just need to understand what the sums of functors are. It will be useful to discuss products of functors at the same time.

To undertand the sums and products of set-valued functors, we will first need to understand the sums and products of sets.

2.2.1 Dependent sums and products of sets

Let I be a set, and let X_i be a set for each $i \in I$. We denote this *I-indexed collection of sets* — a *dependent set* — in the form of a functor as $X \colon I \to \mathbf{Set}$ (where we view the set I as a discrete category) or more classically as $(X_i)_{i \in I}$. Indeed, when a functor $X \colon I \to \mathbf{Set}$ is understood to be a dependent set, we will denote $X(i)$ as X_i for $i \in I$. A single element from one set in the collection would be denoted (i, x) where $x \in X_i$. Choosing an element from every set in the collection would give us a function $i \mapsto x_i$, where each $x_i \in X_i$. This is a sort of function we haven't seen before, at least in this form — its codomain *depends* on the element we are applying it to. Think of a vector field v: to each point p it assignes a tangent vector v_p in the tangent space *at* p. We can write the signature of such a function as

$$f \colon (i \in I) \to X_i.$$

We call this a *dependent function*, since its codomain depends on the element of its domain that we are applying it to.

[1] For those who clamor for polynomials in many variables, we will see in **??** that the multivariable story falls out of the one-variable story.

Definition 2.14 (Dependent sums and products of sets)**.** Let I be a set and $X\colon I \to \mathbf{Set}$ be an I-indexed collection of sets. The *sum* $\sum_{i\in I} X_i$ and *product* $\prod_{i\in I} X_i$ of this collection are the sets

$$\sum_{i\in I} X_i := \{(i,x) \mid i \in I \text{ and } x \in X_i\} \qquad \text{and} \qquad \prod_{i\in I} X_i := \{f\colon (i \in I) \to X_i\}.$$

Example 2.15. If $I = \mathsf{2} = \{1,2\}$ then a collection $X\colon I \to \mathbf{Set}$ is just two sets, say $X_1 = \{a,b,c\}$ and $X_2 = \{c,d\}$. Their sum is the disjoint union

$$\sum_{i\in\mathsf{2}} X_i = X_1 + X_2 = \{(1,a),(1,b),(1,c),(2,c),(2,d)\}.$$

Its cardinality (i.e. the number of elements it contains) will always be the sum of the cardinalities of X_1 and X_2.

Meanwhile, their product is the usual cartesian product

$$\prod_{i\in\mathsf{2}} X_i \cong X_1 \times X_2 = \{(a,c),(a,d),(b,c),(b,d),(c,c),(c,d)\}.$$

Its cardinality will always be the product of the cardinalities of X_1 and X_2.

Exercise 2.16 (Solution here). Let I be a set and let $X_i := \mathsf{1}$ be a one-element set for each $i \in I$.

1. Show that there is an isomorphism of sets $I \cong \sum_{i\in I} \mathsf{1}$.
2. Show that there is an isomorphism of sets $\mathsf{1} \cong \prod_{i\in I} \mathsf{1}$.

As a special case, suppose $I := \varnothing$ and $X\colon \varnothing \to \mathbf{Set}$ is the unique empty collection of sets.

3. Is it true that $X_i = \mathsf{1}$ for each $i \in I$?
4. Show that there is an isomorphism of sets $\mathsf{0} \cong \sum_{i\in\varnothing} X_i$, to justify the statement "the empty sum is $\mathsf{0}$."
5. Show that there is an isomorphism of sets $\mathsf{1} \cong \prod_{i\in\varnothing} X_i$, to justify the statement "the empty product is $\mathsf{1}$." ◊

We'll assume you're already familiar with the following fact, which justifies why we call the constructions in Definition 2.14 sums and products.

Proposition 2.17. Let I be a set and $X\colon I \to \mathbf{Set}$ be an I-indexed collection of sets. Then the sum $\sum_{i\in I} X_i$ is the categorical coproduct of these sets in **Set** (i.e. the colimit of the functor $X\colon I \to \mathbf{Set}$), and the product $\prod_{i\in I} X_i$ is the categorical product of these sets in **Set** (i.e. the limit of the functor $X\colon I \to \mathbf{Set}$).

Exercise 2.18 (Solution here). Let $X\colon I \to \mathbf{Set}$ be a set depending on an $i \in I$. There is a projection function $\pi_1\colon \sum_{i \in I} X_i \to I$ defined by $\pi_1(i, x) = i$.

1. What is the signature of the second projection $\pi_2(i, x) = x$? (Hint: it's a dependent function.)
2. A *section* of a function $r\colon A \to B$ is a function $s\colon B \to A$ such that $s \mathbin{⨟} r = \mathrm{id}_B$. Show that the dependent product is isomorphic to the set of sections of π_1:

$$\prod_{i \in I} X_i \cong \left\{ s\colon I \to \sum_{i \in I} X_i \;\middle|\; s \mathbin{⨟} \pi_1 = \mathrm{id}_I \right\}.$$

◊

A helpful way to think about sum or product sets is by considering what choices must be made to specify an element of such a set. In the following examples, say that we have a dependent set $X\colon I \to \mathbf{Set}$.

Below, we give the instructions for choosing an element of $\sum_{i \in I} X_i$.

To choose an element of $\sum_{i \in I} X_i$:

1. choose an element $i \in I$;
2. choose an element of X_i.

Then the projection π_1 from Exercise 2.18 sends each element of $\sum_{i \in I} X_i$ to the element of $i \in I$ chosen in step 1, while the projection π_2 sends each element of $\sum_{i \in I} X_i$ to the element of X_i chosen in step 2.

Next, we give the instructions for choosing an element of $\prod_{i \in I} X_i$.

To choose an element of $\prod_{i \in I} X_i$:

1. for each element $i \in I$:
   1.1. choose an element of X_i.

Armed with these interpretations, we can tackle more complicated expressions, including those with nested $\sum$'s and $\prod$'s, such as

$$A \coloneqq \sum_{i \in I} \prod_{j \in J(i)} \sum_{k \in K(i,j)} X(i, j, k). \tag{2.19}$$

The instructions for choosing an element of A form a nested list, as follows.

To choose an element of A:

1. choose an element $i \in I$;
2. for each element $j \in J(i)$:
   2.1. choose an element $k \in K(i, j)$;
   2.2. choose an element of $X(i, j, k)$.

Note that the choice of $k \in K(i, j)$ can depend on i and j; it must be able to, because different values of i and j may lead to different sets $K(i, j)$.

By describing A like this, it is clear that each $a \in A$ can be projected to an element $\pi_1(a) \in I$ from step 1 and a dependent function $\pi_2(a)$ from step 2. This dependent function in turn sends each $j \in J(i)$ to a pair that can be projected to an element $\pi_1(\pi_2(a)(j)) \in K(i, j)$ from step 2.1 and an element $\pi_2(\pi_2(a)(j)) \in X(i, j, k)$ from step 2.2.

Example 2.20. Let $I = \{1, 2\}$; let $J(1) = \{j\}$ and $J(2) \coloneqq \{j, j'\}$; let $K(1, j) \coloneqq \{k_1, k_2\}$, $K(2, j) \coloneqq \{k_1\}$, and $K(2, j') \coloneqq \{k'\}$; and let $X(i, j, k) = \{x, y\}$ for all i, j, k. Now the formula

$$\sum_{i \in I} \prod_{j \in J(i)} \sum_{k \in K(i,j)} X(i, j, k)$$

from (2.19) has been given meaning as an actual set. Here is a list of all eight of its elements:

$$\left\{\begin{array}{cccc} \big(1, j \mapsto (k_1, x)\big), & \big(1, j \mapsto (k_1, y)\big), & \big(1, j \mapsto (k_2, x)\big), & \big(1, j \mapsto (k_2, y)\big), \\ & \big(2, j \mapsto (k_1, x), j' \mapsto (k', x)\big), & \big(2, j \mapsto (k_1, x), j' \mapsto (k', y)\big), & \\ & \big(2, j \mapsto (k_1, y), j' \mapsto (k', x)\big), & \big(2, j \mapsto (k_1, y), j' \mapsto (k', y)\big) & \end{array}\right\}$$

In each case, we first chose an element $i \in I$, either 1 or 2. Then for each $j \in J(i)$ we chose an element $k \in K(i, j)$; then we concluded by choosing an element of $X(i, j, k)$.

Exercise 2.21 (Solution here). Consider the set

$$B \coloneqq \prod_{i \in I} \sum_{j \in J(i)} \prod_{k \in K(i,j)} X(i, j, k). \tag{2.22}$$

1. Give the instructions for choosing an element of B as a nested list, like we did for A just below (2.19).
2. With I, J, K, and X as in Example 2.20, how many elements are in B?
3. Write out three of these elements in the style of Example 2.20. ◊

From here on out, we won't write out the full sequence of nested instructions corresponding to every dependent sum or product; we'll assume you're able to read them for yourself.

2.2.2 Expanding products of sums

We will often encounter sums of dependent sets nested within products, as in (2.22). The following proposition helps us work with these; it is sometimes called the *type-theoretic axiom of choice*. Though we will take the time to walk you through its proof, it is almost trivial, once you understand what it's saying. For one thing, it is exactly how you would expect the distributive property of multiplication over addition to work, following the same sort of process that you would use to multiply multi-digit numbers from grade school or polynomials from high school algebra; for another, once the statement is written in Agda, its proof is one short line of Agda code.

Proposition 2.23 (Pushing $\prod$ past $\sum$)**.** For any set I, sets $(J(i))_{i\in I}$, and sets $(X(i,j))_{i\in I, j\in J(i)}$, we have a natural isomorphism

$$\prod_{i\in I}\sum_{j\in J(i)} X(i,j) \cong \sum_{\bar{j}\in\prod_{i\in I} J(i)}\;\prod_{i\in I} X(i,\bar{j}(i)).^{a} \tag{2.24}$$

[a] We draw a bar over j in $\bar{j}$ to remind ourselves that $\bar{j}$ is no longer just an index but a (dependent) function.

Proof. We'll do this the old-fashioned way: by giving a map from left to right, a map from right to left, and a proof that the two maps are mutually inverse.

First, let's go from left to right. An element of the set on the left is a dependent function $f\colon (i\in I)\to\sum_{j\in J(i)} X(i,j)$, which we can compose with projections from its codomain to yield $\pi_1(f(i))\in J(i)$ and $\pi_2(f(i))\in X(i,\pi_1(f(i)))$ for every $i\in I$. We can then form the following pair in the right hand set:

$$(i\mapsto\pi_1(f(i)),\, i\mapsto\pi_2(f(i))).$$

Now let's go from right to left. An element of the set on the right is a pair of dependent functions, $\bar{j}\colon (i\in I)\to J(i)$ and $g\colon (i\in I)\to X(i,\bar{j}(i))$. We then get an element of the set on the left as follows:

$$i\mapsto(\bar{j}(i), g(i)).$$

Now, we just need to check that a round trip takes us back where we were. If we start on the right from $(\bar{j}, g)$, our round trip gives us the pair

$$(i\mapsto\pi_1(\bar{j}(i), g(i)),\, i\mapsto\pi_2(\bar{j}(i), g(i))).$$

But $\pi_1(\bar{j}(i), g(i)) = \bar{j}(i)$ and $\pi_2(\bar{j}(i), g(i)) = g(i)$ by definition, so we're back where we started. On the other hand, starting on the left from f gives us the function

$$i\mapsto(\pi_1(f(i)), \pi_2(f(i))).$$

But again, since $f(i)$ is a pair whose components are $\pi_1(f(i))$ and $\pi_2(f(i))$, we're back where we started. □

When $J(i) = J$ does not depend on $i\in I$, the formula in (2.24) becomes much easier.

Corollary 2.25. For any set I, set J, and sets $(X(i,j))_{i\in I, j\in J}$, we have a natural isomorphism

$$\prod_{i\in I}\sum_{j\in J} X(i,j) \cong \sum_{\bar{j}\colon I\to J}\prod_{i\in I} X(i,\bar{j}(i)). \tag{2.26}$$

Proof. Just take $J(i) = J$ for all $i \in I$ in (2.24). Note that dependent functions $\bar{j}$ in $\prod_{i \in I} J(i)$ then become standard functions $\bar{j} \colon I \to J$. □

It turns out that being able to push $\prod$ past $\sum$ as in (2.24) is not a property that is unique to sets. In general, we refer to a category having this property as follows.

Definition 2.27. A category $\mathcal{C}$ with all small products and coproducts is *completely distributive* if products distribute over coproducts as in (2.24); that is, for any set I, sets $(J(i))_{i \in I}$, and objects $(X(i,j))_{i \in I, j \in J(i)}$ from $\mathcal{C}$, we have a natural isomorphism

$$\prod_{i \in I} \sum_{j \in J(i)} X(i,j) \cong \sum_{\bar{j} \in \prod_{i \in I} J(i)} \prod_{i \in I} X(i, \bar{j}(i)). \tag{2.28}$$

So Proposition 2.23 states that **Set** is completely distributive. We'll see shortly that **Poly** is, too. Of course, Corollary 2.25 generalizes to all completely distributive categories as well.

Corollary 2.29. Let $\mathcal{C}$ be a completely distributive category. For any set I, set J, and objects $(X(i,j))_{i \in I, j \in J}$ from $\mathcal{C}$, we have a natural isomorphism

$$\prod_{i \in I} \sum_{j \in J} X(i,j) \cong \sum_{\bar{j} \colon I \to J} \prod_{i \in I} X(i, \bar{j}(i)). \tag{2.30}$$

Proof. Again, just take $J(i) = J$ for all $i \in I$ in (2.28). □

Exercise 2.31 (Solution here). Let $\mathcal{C}$ be a completely distributive category. How is the usual distributive law,

$$X(Y+Z) \cong XY + XZ$$

for $X, Y, Z \in \mathcal{C}$, a special case of (2.28)? ◊

Throughout this book, e.g. in Exercise 2.60, you'll often see an expression consisting of alternating products and sums. Using (2.28), you can always write such an expression as a sum of products, in which every $\sum$ appears before every $\prod$ (i.e. in "disjunctive normal form"). This is analogous to how products of sums in high school algebra can always be expanded into sums of products via the distributive property.

Exercise 2.32 (Solution here). Let $I, (J(i))_{i \in I}, (K(i,j))_{(i,j) \in IJ}$ be sets, and for each $(i,j,k) \in IJK$, let $X(i,j,k)$ be an object in a completely distributive category.

1. Rewrite

$$\sum_{i \in I} \prod_{j \in J(i)} \sum_{k \in K(i,j)} X(i,j,k)$$

so that every $\sum$ appears before every $\prod$.

2. Rewrite
$$\prod_{i \in I} \sum_{j \in J(i)} \prod_{k \in K(i,j)} X(i,j,k)$$
so that every $\sum$ appears before every $\prod$.
3. Rewrite
$$\prod_{i \in I} \prod_{j \in J(i)} \sum_{k \in K(i,j)} X(i,j,k)$$
so that every $\sum$ appears before every $\prod$. ◊

Now that we have defined dependent sums and products of sets, we are ready to tackle dependent sums and products of set-valued functors.

2.2.3 Dependent sums and products of functors **Set** → **Set**

Remember: our goal is to define polynomial functors, e.g. $y^2 + 2y + 1$, and the maps between them. Since y^2, y, and 1 are functors, we just need to interpret sums of functors **Set** → **Set**. But we might as well examine products of functors at the same time, because they'll very much come in handy. And both these concepts generalize to limits and colimits in $[\mathbf{Set}, \mathbf{Set}]$.

Proposition 2.33. The category $[\mathbf{Set}, \mathbf{Set}]$ has all small limits and colimits, and they are computed pointwise. That is, given a small category $\mathcal{J}$ and a functor $F \colon \mathcal{J} \to [\mathbf{Set}, \mathbf{Set}]$, for all $X \in \mathbf{Set}$, the limit and colimit of F satisfy isomorphisms

$$\left(\lim_{j \in \mathcal{J}} F(j)\right)(X) \cong \lim_{j \in \mathcal{J}} (F(j)(X)) \qquad \text{and} \qquad \left(\operatorname*{colim}_{j \in \mathcal{J}} F(j)\right)(X) \cong \operatorname*{colim}_{j \in \mathcal{J}} (F(j)(X))$$

natural in X.

Proof. This is a special case of a more general fact, where $[\mathbf{Set}, \mathbf{Set}]$ is replaced by an arbitrary functor category $[\mathcal{D}, \mathcal{C}]$, where $\mathcal{C}$ is a category that (like **Set**) has limits and colimits; see [MM92, page 22 – 23, displays (24) and (25)]. □

Example 2.34 (Dependent sums and products of functors **Set** → **Set**). For any two functors $F, G \colon \mathbf{Set} \to \mathbf{Set}$, let

$$(F + G) \colon \mathbf{Set} \to \mathbf{Set} \qquad \text{and} \qquad (F \times G) \colon \mathbf{Set} \to \mathbf{Set}$$

respectively denote the categorical *sum* (i.e. *coproduct*) and *product* of F and G in $[\mathbf{Set}, \mathbf{Set}]$.[a] Then by Proposition 2.33, for each $X \in \mathbf{Set}$,

$$(F + G)(X) \cong F(X) + G(X) \qquad \text{and} \qquad (F \times G)(X) \cong F(X) \times G(X).$$

More generally, for any set I and functors $(F_i)_{i\in I}$, let

$$\sum_{i\in I} F_i \colon \mathbf{Set} \to \mathbf{Set} \qquad \text{and} \qquad \prod_{i\in I} F_i \colon \mathbf{Set} \to \mathbf{Set}$$

respectively denote the categorical *sum* (i.e. *coproduct*) and *product* of the functors $(F_i)_{i\in I}$. Then by Proposition 2.33, for each $X \in \mathbf{Set}$,

$$\left(\sum_{i\in I} F_i\right)(X) \cong \sum_{i\in I} F_i(X) \qquad \text{and} \qquad \left(\prod_{i\in I} F_i\right)(X) \cong \prod_{i\in I} F_i(X).$$

Given a set $I \in \mathbf{Set}$, we will also use I to denote the constant functor that assigns I to each $X \in \mathbf{Set}$. In particular, we denote by $\mathsf{0}, \mathsf{1} \colon \mathbf{Set} \to \mathbf{Set}$ the constant functors that respectively assign $\mathsf{0}$ and $\mathsf{1}$ to each $X \in \mathbf{Set}$. As the set $\mathsf{0}$ (resp. $\mathsf{1}$) is the initial (resp. terminal) object in $\mathbf{Set}$, Proposition 2.33 tells us that the constant functor $\mathsf{0}$ (resp. $\mathsf{1}$) is the is the initial (resp. terminal) object in $[\mathbf{Set}, \mathbf{Set}]$.

[a] We may also denote the product functor $F \times G$ by FG.

Proposition 2.35. The category $[\mathbf{Set}, \mathbf{Set}]$ is completely distributive.

Proof. This follows directly from the fact that **Set** itself is completely distributive (Proposition 2.23) and the fact that sums and products in $[\mathbf{Set}, \mathbf{Set}]$ are computed pointwise (Proposition 2.33 and, in particular, Example 2.34). □

Exercise 2.36 (Solution here).

1. Show that for a set $I \in \mathbf{Set}$ and a functor $F \colon \mathbf{Set} \to \mathbf{Set}$, the sum of I copies of F is isomorphic to the product of the constant functor I and F:

$$\sum_{i\in I} F \cong IF.$$

(This is analogous to the fact from basic arithmetic that adding up $n \in \mathbb{N}$ copies of number is equal to multiplying that same number by n.)

2. So does $\mathsf{2}y$ denote $\mathsf{2} \times y$ or $y + y$, or does it not matter for some reason? ◊

Exercise 2.37 (Solution here).

1. Show that for a set $I \in \mathbf{Set}$, the product of I copies of the identity functor $y \colon \mathbf{Set} \to \mathbf{Set}$ is isomorphic to the functor $y^I \colon \mathbf{Set} \to \mathbf{Set}$ represented by I:

$$\prod_{i\in I} y \cong y^I.$$

(This is analogous to the fact from basic arithmetic that multiplying n copies of a number together is equal to raising that same number to the power of n.)

2. Show that the product of I copies of a representable functor $y^A \colon \mathbf{Set} \to \mathbf{Set}$ for some $A \in \mathbf{Set}$ is isomorphic to the functor $y^{IA} \colon \mathbf{Set} \to \mathbf{Set}$ represented by the product set IA:
$$\prod_{i \in I} y^A \cong y^{IA}.$$

◊

Remark 2.38. From here on out, given $I \in \mathbf{Set}$ and a functor $F \colon \mathbf{Set} \to \mathbf{Set}$, we define
$$F^I \coloneqq \prod_{i \in I} F.$$

The exercise above shows that this notation does not conflict with the way we write representable functors as powers of the identity functor y. The exercise also shows how the power of a representable functor can be simplified to a single representable functor.

We've finally arrived: we can define polynomial functors!

2.2.4 What is a polynomial functor?

Definition 2.39 (Polynomial functors)**.** A *polynomial functor* (or simply a *polynomial*) is a functor $p \colon \mathbf{Set} \to \mathbf{Set}$ such that there exists a set I, sets $(p[i])_{i \in I}$, and an isomorphism
$$p \cong \sum_{i \in I} y^{p[i]}$$
to a sum of representables.

So (up to isomorphism), a polynomial functor is just a sum of representables.

Remark 2.40. Given sets $I, A \in \mathbf{Set}$, it follows from Exercise 2.36 that we have an isomorphism of polynomials
$$\sum_{i \in I} y^A \cong I y^A.$$
So when we write down a polynomial, we will often combine identical representable summands y^A by writing them in the form Iy^A. In particular, the constant functor 1 is a representable functor ($1 \cong y^0$), so every constant functor I is a polynomial functor: $I \cong \sum_{i \in I} 1$.

Example 2.41. Consider the polynomial $p \coloneqq y^2 + 2y + 1$. It denotes a functor $\mathbf{Set} \to \mathbf{Set}$; what does this functor do to the set $X \coloneqq \{a, b\}$? To be very precise and pedantic, let's say
$$I \coloneqq 4 \qquad \text{and} \qquad p[1] \coloneqq 2, \quad p[2] \coloneqq 1, \quad p[3] \coloneqq 1, \quad p[4] \coloneqq 0$$

so that $p \cong \sum_{i \in I} y^{p[i]}$. Now we have

$$p(X) \cong \{(1,a,a),(1,a,b),(1,b,a),(1,b,b),(2,a),(2,b),(3,a),(3,b),(4)\}.$$

It has $(2^2+2+2+1)$-many, i.e. 9, elements. The representable summand y^A throws in all A-tuples from X, but it's indexed by the name of the summand. In particular if $A = 0$ then it just records the empty tuple at that summand.

In general, a polynomial $p \coloneqq \sum_{i \in I} y^{p[i]}$ applied to a set X, expanded as

$$\sum_{i \in I} X^p[i],$$

can be thought of as the set of all pairs comprised of an element of I and a $p[i]$-tuple of elements of X. Alternatively, we can think of each pair as an element of I and a function $p[i] \to XS$.

Exercise 2.42 (Solution here). In the pedantic style of Example 2.41, write out all the elements of $p(X)$ for p and X as follows:

1. $p \coloneqq y^3$ and $X \coloneqq \{4,9\}$.
2. $p \coloneqq 3y^2 + 1$ and $X \coloneqq \{a\}$.
3. $p \coloneqq 0$ and $X \coloneqq \mathbb{N}$.
4. $p \coloneqq 4$ and $X \coloneqq \mathbb{N}$.
5. $p \coloneqq y$ and $X \coloneqq \mathbb{N}$. ◊

Proposition 2.43. Let $p \coloneqq \sum_{i \in I} y^{p[i]}$ be an arbitrary polynomial functor. Then $I \cong p(1)$, so there is an isomorphism of functors

$$p \cong \sum_{i \in p(1)} y^{p[i]}. \tag{2.44}$$

Proof. We need to show that $I \cong p(1)$; the latter claim follows directly. In Exercise 2.16 it was shown that $I \cong \sum_{i \in I} 1$, so we just need to show that $(y^{p[i]})(1) \cong 1$ for every $i \in I$. But $1^{p[i]} \cong 1$ because there is a unique function $p[i] \to 1$ for any $p[i]$. □

We can draw an analogy between Proposition 2.43 and evaluating $p(1)$ for a polynomial p from high school algebra, which yields the sum of the coefficients of p. The notation in (2.44) will be how we denote arbitrary polynomials from now on.

Exercise 2.45 (Solution here). We saw in Proposition 2.43 that for any polynomial p, e.g. $p \coloneqq y^3 + 3y^2 + 4$, the set $p(1)$ gives back the set of summands, in this case 8.

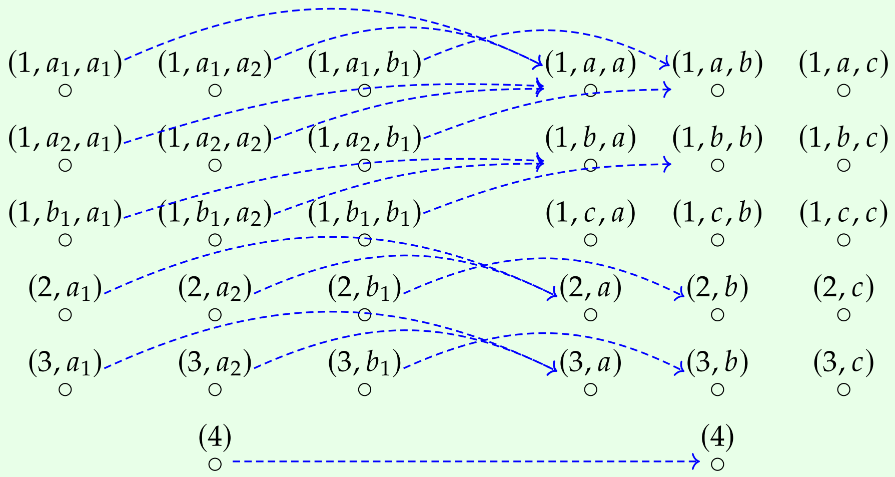
What does $p(0)$ give you? ◊

As a functor **Set** → **Set**, a polynomial should be able to act not only on sets but also on functions. Below, we explain how.

Proposition 2.46. Let p be an arbitrary polynomial functor (our notation allows us to write it as $p \cong \sum_{i \in p(1)} y^{p[i]}$), and let $f\colon X \to Y$ be an arbitrary function. Then $p(f)\colon p(X) \to q(X)$ sends each $(i, g) \in p(X)$, with $i \in p(1)$ and $g\colon p[i] \to X$, to $(i, g \operatorname{⨟} f)$ in $p(Y)$.

Proof. For each $i \in p(1)$, by Definition 2.1, the functor $y^{p[i]}$ sends f to the function $f^{p[i]}\colon X^{p[i]} \to Y^{p[i]}$, the one that sends every $g\colon p[i] \to X$ to $g \operatorname{⨟} f\colon p[i] \to Y$. So the sum of these functors over $i \in p(1)$ sends each $(i, g) \in p(X)$ to $(i, f^{p[i]}(g)) = (i, g \operatorname{⨟} f) \in p(Y)$, as desired. □

Example 2.47. Suppose $p \coloneqq y^2 + 2y + 1$. Let $X \coloneqq \{a_1, a_2, b_1\}$ and $Y \coloneqq \{a, b, c\}$, and let $f\colon X \to Y$ be the function sending $a_1, a_2 \mapsto a$ and $b_1 \mapsto b$. The induced function $p(f)\colon p(X) \to p(Y)$, according to Proposition 2.46, is shown below:



Exercise 2.48 (Solution here). Let $p \coloneqq y^2 + y$. Choose a function $f\colon 1 \to 2$ and write out the induced function $p(f)\colon p(1) \to p(2)$. ◊

2.3 Morphisms between polynomial functors

Before we define the category **Poly** of polynomial functors, we notice that polynomial functors already live inside a category, namely the category [**Set**, **Set**] of functors **Set** → **Set**, whose morphisms are natural transformations. This leads to a very natural (if you will) definition of morphisms between polynomial functors, from which we can derive a category of polynomial functors for free.

Definition 2.49 (Polynomial morphisms, **Poly**). Given polynomial functors p and q, a *morphism of polynomial functors* (or a *polynomial morphism*) is a natural transformation $p \to q$. Then **Poly** is the category whose objects are polynomial functors and whose morphisms are polynomial morphisms.

In other words, **Poly** is the full subcategory of $[\mathbf{Set}, \mathbf{Set}]$ spanned by the polynomial functors: we take the category $[\mathbf{Set}, \mathbf{Set}]$, throw out all the objects that are not polynomials, but keep all the same morphisms between any two polynomial functors.

2.3.1 Coproducts of polynomials

Since polynomial functors are defined as arbitrary sums of representables, coproducts in **Poly** are quite easy to understand.

Proposition 2.50. The category **Poly** has arbitrary coproducts, coinciding with coproducts in $[\mathbf{Set}, \mathbf{Set}]$ given by the operation $\sum_{i \in I}$.

Proof. By Example 2.34, the category $[\mathbf{Set}, \mathbf{Set}]$ has arbitrary coproducts given by $\sum_{i \in I}$. The full subcategory inclusion $\mathbf{Poly} \to [\mathbf{Set}, \mathbf{Set}]$ reflects these coproducts, and by definition **Poly** is closed under the operation $\sum_{i \in I}$. □

Explicitly, given polynomials $(p_i)_{i \in I}$, their coproduct is

$$\sum_{i \in I} p_i \cong \sum_{i \in I} \sum_{j \in p_i(1)} y^{p_i[j]} \cong \sum_{(i,j) \in \sum_{i \in I} p_i(1)} y^{p_i[j]}, \tag{2.51}$$

which coincides with our notion of sums of functors $\mathbf{Set} \to \mathbf{Set}$ from Example 2.34. Binary coproducts are thus given by binary sums of functors, denoted by $+$, while the initial object of **Poly** is the constant polynomial 0.

In particular, (2.51) implies that for any polynomials p and q, their coproduct $p + q$ is given as follows. The position-set of $p + q$ is the coproduct of sets $p(1) + q(1)$. At position $(1, i) \in p(1) + q(1)$ with $i \in p(1)$, the directions of $p + q$ are just the $p[i]$-directions; at position $(2, j) \in p(1) + q(1)$ with $j \in q(1)$, the directions of $p + q$ are just the $q[j]$-directions.

2.3.2 Polynomial morphisms, concretely

A natural transformation between polynomials $p \to q$ consists of a function $p(X) \to q(X)$ for every set $X \in \mathbf{Set}$ such that the naturality squares commute. That's a lot of data to keep track of! Fortunately, there is a much simpler way to think about these polynomial morphisms, which we will discover with some help from our old friend, the Yoneda lemma.

Exercise 2.52 (Solution here). Given a set S and a polynomial q, show that a polynomial morphism $y^S \to q$ can be identified with an element of the set $q(S)$. That is, there is an isomorphism

$$\mathbf{Poly}(y^S, q) \cong q(S).$$

Moreover, show that this isomorphism is natural in both S and q. Hint: Use the Yoneda lemma (Lemma 2.10). ◊

Before we present our alternative characterization of polynomial morphisms, recall that every polynomial $p := \sum_{i \in p(1)} y^{p[i]}$ is uniquely associated to a dependent set, $(p[i])_{i \in p(1)}$, which we call its *arena*, as in (1.3). Alternatively, we could write such a dependent set as a functor $p[-] \colon p(1) \to \mathbf{Set}$, where we view the set $p(1)$ as a discrete category. Below, we make use of this functor notation to express the arenas of polynomials.

Proposition 2.53. Let $p := \sum_{i \in p(1)} y^{p[i]}$ and $q := \sum_{j \in q(1)} y^{q[j]}$ be polynomials. Then we have an isomorphism

$$\mathbf{Poly}(p, q) \cong \prod_{i \in p(1)} \sum_{j \in q(1)} p[i]^{q[j]} \tag{2.54}$$

natural in p and q. In other words, a morphism $p \to q$ can be identified with a pair $(f_1, f^\sharp)$

$$\begin{tikzcd} p(1) \arrow[rr, "f_1"] \arrow[dr, "{p[-]}"'] & \arrow[d, phantom, "\overset{f^\sharp}{\Leftarrow}"] & q(1) \arrow[dl, "{q[-]}"] \\ & \mathbf{Set} & \end{tikzcd} \tag{2.55}$$

where $f_1 \colon p(1) \to q(1)$ is a function (or functor between discrete categories) and $f^\sharp \colon q[f_1(-)] \to p[-]$ is a natural transformation: for each $i \in p(1)$ with $j := f_1(i)$, there is a function $f_i^\sharp \colon q[j] \to p[i]$.

Proof. By the universal property of the coproduct, we have a natural isomorphism

$$\mathbf{Poly}\left(\sum_{i \in p(1)} y^{p[i]}, q\right) \cong \prod_{i \in p(1)} \mathbf{Poly}(y^{p[i]}, q),$$

so applying Exercise 2.52 (i.e. the Yoneda lemma) and unraveling the definitions of p and q yields (2.54).

The expression on the right hand side of (2.54) is the set of dependent functions $f \colon (i \in p(1)) \to \sum_{j \in q(1)} p[i]^{q[j]}$, each of which is uniquely determined by its components: $\pi_1 \circ f$, which sends $i \in p(1)$ to $\pi_1(f(i)) \in q(1)$, and $\pi_2 \circ f$, which sends $i \in p(1)$ with $j := \pi_1(f(i))$ to an element of $p[i]^{q[j]}$, i.e. a function $q[j] \to p[i]$. These can be identified respectively with a (non-dependent) function $f_1 := \pi_1 \circ f$ from $p(1) \to q(1)$ and a natural transformation $f^\sharp \colon q[f_1(-)] \to p[-]$. □

We have now greatly simplified our characterization of polynomial morphisms $f\colon p \to q$: rather than infinitely many functions satisfying infinitely many naturality conditions, they can be specified simply as a function $f_1\colon p(1) \to q(1)$ and, for each $i \in p(1)$, a function $f_i^\sharp\colon q[f_1(i)] \to p[i]$, without any additional restrictions.

Here is where we begin to see the advantages of viewing polynomials as arenas. As a reminder, from the arena perspective, we call the elements of $p(1)$ the *positions* of p, and for each position $i \in p(1)$, we call the elements of $p[i]$ the *directions* of p at i. We can see that our characterization of a polynomial morphism can be written entirely in the language of positions and directions: when polynomials p and q are viewed as arenas, a morphism $f\colon p \to q$ consists of a "forwards" *on-positions* function f_1 from the position-set of p to the position-set of q, along with, for every position i of p, a "backwards" *on-directions* function $f_i^\sharp$ from the direction-set of q at $f_1(i)$ to the direction-set of p at i.

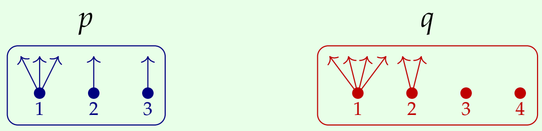
When we wish to emphasize the arena perspective, we will call the data of $(f_1, f^\sharp)$ a *morphism of arenas*, an *arena morphism*, or a *lens* between p and q (see Section 2.3.3); but since Proposition 2.53 tells us that polynomial morphisms and arena morphisms carry the same data, we may use these terms interchangeably.

Here is how you would implement such a morphism in Agda:

```
record Lens (p : Arena) (q : Arena) : Set where
  field
    onPos : (pos p) -> (pos q)
    onDir : (i : pos p) -> dir q (onPos i) -> dir p i
```

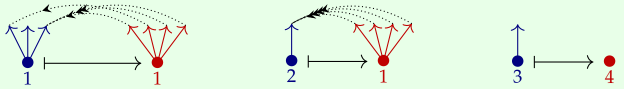
This forwards on-positions/backwards on-directions formulation may still seem a little complicated, so here is some decision-making intuition. Recall from Section 1.3 that we may view each position of an arena as a decision and the directions at that position as the options available for that decision. The morphisms $f\colon p \to q$ are then the ways to *delegate* p's decisions to q. Every one of p's decisions, say $i \in p(1)$, is passed forward to a decision $j \in q(1)$ for q to make, and every choice $d \in q[j]$ that q could make among its options is passed back as some choice $c \in p[i]$ among p's options.

Example 2.56. Let $p := y^3 + 2y$ and $q := y^4 + y^2 + 2$. Here they are, depicted as corolla forests:



To give a map of polynomials $p \to q$, one sends each position $i \in p(1)$ of p to a position $j \in q(1)$ of q, then sends each direction in $q[j]$ back to one in $p[i]$.

How many ways are there to do this? Before answering this, let's just pick one.



This represents one morphism $p \to q$.

So how many different morphisms are there from p to q? The first p-position can be sent to any q-position: 1, 2, 3, or 4. Sending it to 1 requires choosing how each of the four options ($q[1] = 4$) are to be assigned one of $p[1] = 3$ options; there are 3^4 ways to do this. Similarly, we can calculate the remaining ways to handle the first p-position, then add them up: there are $3^4 + 3^2 + 3^0 + 3^0 = 92$ ways total.

The second p-position can also be sent to 1, 2, 3, or 4, before sending back directions; there are $1^4 + 1^2 + 1^0 + 1^0 = 4$ ways to do this. Similarly there are four ways to send the third p-position to a q-position and send back directions.

In total, there are $92 \cdot 4 \cdot 4 = 1472$ morphisms $p \to q$.

Unsurprisingly, this is exactly what is given by (2.54):

$$\begin{aligned}
|\mathbf{Poly}(p, q)| &= \prod_{i \in p(1)} |q(p[i])| \\
&= \prod_{i \in p(1)} |p[i]|^4 + |p[i]|^2 + 2 \\
&= (3^4 + 3^2 + 2)(1^4 + 1^2 + 2)^2 \\
&= 92 \cdot 4^2 = 1472.
\end{aligned}$$

Exercise 2.57 (Solution here).

1. Draw the corolla forests associated to $p := y^3 + y + 1$, $q := y^2 + y^2 + 2$, and $r := y^3$.
2. Give an example of a morphism $p \to q$ and draw it as we did in Example 2.56.
3. Explain your morphism intuitively as a delegation of decisions.
4. Explain in those terms why there can't be any morphisms $p \to r$. ◊

Exercise 2.58 (Solution here). For any polynomial p and set A, e.g. $A = 2$, the Yoneda lemma gives an isomorphism $p(A) \cong \mathbf{Poly}(y^A, p)$.

1. Choose a polynomial p and draw both y^2 and p as corolla forests.
2. Count all the maps $y^2 \to p$. How many are there?
3. Is the previous answer equal to $p(2)$? ◊

Exercise 2.59 (Solution here). For each of the following polynomials p, q, compute the number of morphisms $p \to q$.

1. $p = y^3, \quad q = y^4$.
2. $p = y^3 + 1, \quad q = y^4$.
3. $p = y^3 + 1, \quad q = y^4 + 1$.
4. $p = 4y^3 + 3y^2 + y, \quad q = y$.
5. $p = 4y^3, \quad q = 3y$. ◊

Exercise 2.60 (Solution here).

1. Show that the following are isomorphic:

$$\mathbf{Poly}(p, q) \cong^? \prod_{i \in p(1)} \sum_{j \in q(1)} \prod_{d \in q[j]} \sum_{c \in p[i]} 1 \tag{2.61}$$

2. Show that the following are isomorphic:

$$\mathbf{Poly}(p, q) \cong^? \sum_{f_1 \colon p(1) \to q(1)} \prod_{j \in q(1)} \mathbf{Set}\left(q[j], \prod_{\substack{i \in p(1), \\ f_1(i) = j}} p[i]\right) \tag{2.62}$$

3. Describe in the language of decision-making how any element of the right-hand side of (2.62) gives a way of delegating decisions from p to q. ◊

Exercise 2.63 (Solution here). Use (2.51) and (2.54) to verify that

$$\mathbf{Poly}\left(\sum_{i \in I} p_i, q\right) \cong \prod_{i \in I} \mathbf{Poly}(p_i, q)$$

for all polynomials $(p_i)_{i \in I}$ and q, as expected from the universal property of coproducts. ◊

Example 2.64 (Derivatives). The *derivative* of a polynomial p, denoted $\dot{p}$, is defined as follows:

$$\dot{p} \coloneqq \sum_{i \in p(1)} \sum_{d \in p[i]} y^{p[i] - \{d\}}.$$

For example, if $p \coloneqq y^{\{U,V,W\}} + \{A, B\}y^{\{X\}}$ then

$$\dot{p} = \{U\}y^{\{V,W\}} + \{V\}y^{\{U,W\}} + \{W\}y^{\{U,V\}} + \{(A, X), (B, X)\}y^0.$$

Up to isomorphism $p \cong y^3 + 2y$ and $\dot{p} \cong 3y^2 + 2$. Unsurprisingly, this coincides with the familiar notion of derivatives of polynomials from calculus.

Thus we get a canonical map $\dot{p}y \to p$, because we have an isomorphism

$$\dot{p}y \cong \sum_{i \in p(1)} \sum_{d \in p[i]} y^{p[i]}.$$

This natural transformation comes up in computer science in the context of "plugging in to one-hole contexts"; we will not explore that here, but see **[mcbride]** and **[abbot2003derivatives]** for more info.

A morphism $f : p \to \dot{q}$ can be interpreted as something like an arena morphism from p to q, except that each p-position explicitly selects a direction of q to remain unassigned. More precisely, for each $i \in p(1)$ we have $f_1(i) = (j, d) \in \sum_{j \in q(1)} q[j]$, i.e. a choice of position j of q, as usual, together with a chosen direction $d \in q[j]$. Then every $q[j]$-direction *other than* d is sent back to a $p[i]$-direction.

Exercise 2.65 (Solution here). Show that $\dot{p}(1)$ is isomorphic to the set of all directions of p (i.e. the union of all direction-sets of p), so there is a canonical function $\pi_p : \dot{p}(1) \to p(1)$ that sends each direction d of p to the position i of p for which $d \in p[i]$. ◊

Exercise 2.66 (Solution here). The derivative is not very well-behaved category-theoretically. However, it is intriguing. Below $p, q \in$ **Poly**.

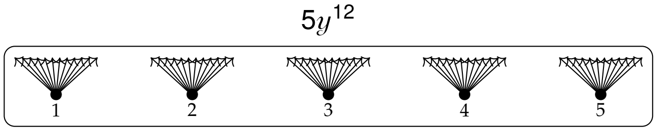
1. Explain the canonical map $\dot{p}y \to p$ from Example 2.64 in more detail.
2. Is there always a canonical map $p \to \dot{p}$?
3. Is there always a canonical map $\dot{p} \to p$?
4. If given a map $p \to q$, does one get a map $\dot{p} \to \dot{q}$?
5. We will define the binary operations $\otimes$ and $[-,-]$ on **Poly** later on in (2.89) and (3.77), and in Exercise 3.82, you will be able to use Exercise 3.80 to deduce that

$$[p, y] \otimes p \cong \sum_{f \in \prod_{i \in p(1)} p[i]} \sum_{i \in p(1)} y^{p(1) \times p[i]}, \tag{2.67}$$

   Is there always a canonical map $[p, y] \otimes p \to \dot{p}$?
6. When talking to someone who explains maps $p \to \dot{q}$ in terms of "unassigned directions," how might you describe what is modeled by a map $py \to q$? ◊

2.3.3 Arena morphisms as lenses

Monomials are special polynomials: those of the form Ay^B for sets A, B. Here's a picture of $5y^{12}$:



The formula for morphisms between these is particularly simple:

$$\begin{aligned}\mathbf{Poly}\left(A_1y^{B_1},\, A_2y^{B_2}\right) &\cong \prod_{a\in A_1}\sum_{a'\in A_2} B_1^{B_2} \qquad (2.54)\\ &\cong \mathbf{Set}(A_1, A_2\times B_1^{B_2})\\ &\cong \mathbf{Set}(A_1,A_2)\times\mathbf{Set}(A_1\times B_2, B_1).\end{aligned}$$

It says that to give a morphism from one monomial to another, you just need to give two (non-dependent!) functions. Let's rewrite it to make those two functions explicit—they are the familiar on-positions and on-directions functions:

$$\mathbf{Poly}\left(A_1y^{B_1},\, A_2y^{B_2}\right) \cong \left\{(f_1, f^\sharp)\;\middle|\; \begin{array}{l} f_1\colon A_1\to A_2\\ f^\sharp\colon A_1\times B_2\to B_1\end{array}\right\}.$$

Ordinarily, $f^\sharp$ is more involved: its type depends on the directions at each position of the domain and its image position via f_1. But for monomials, every position has the same set of directions, so $f^\sharp$ is just a standard function.

The monomials in **Poly** and the morphisms between them form a full subcategory of **Poly**, and it has been called the *the category of bimorphic lenses* [Hed18]. It comes up in functional programming. The functions $f_1, f^\sharp$ corresponding to a morphism $f\colon A_1y^{B_1}\to A_2y^{B_2}$ are given special names:

$$\begin{aligned}\text{get} &:= f_1\colon A_1\to A_2\\ \text{set} &:= f^\sharp\colon A_1\times B_2\to B_1\end{aligned}\qquad (2.68)$$

The idea is that each position $a\in A_1$ of $A_1y^{B_1}$ "gets" a position $f_1(a)\in A_2$ of $A_2y^{B_2}$, and given $a\in A_1$, every direction at $f_1(a)$ in B_2 "sets" a direction back at a in B_1.

So an arena morphism between two monomials is just a bimorphic lens, or a lens for short. Then an arena morphism between any two arenas is a kind of generalized lens. In fact, henceforth we will refer to arena morphisms simply as *lenses*.

Example 2.69. Consider the monomial Sy^S. As an arena, its position-set is S, and its direction-set at each position $s\in S$ is again just S. In the language of decision-making, each $s\in S$ is a decision where the options you have to choose from are always just the decisions in S again. Notice that there is a natural way to string together a series of such decisions into a cycle: at each step, you start at some element of S, and the option you select is the element of S that you will move to next. We will formalize this idea in **??**.

A lens (get, set): $Sy^S\to Ty^T$ is as usual a way to delegate decisions of Sy^S to decisions of Ty^T. When you need to make a decision at $s\in S$, you ask your friend at get$(s)\in T$ for help. If your friend selects option $t\in T$, then you know to select option set$(s,t)\in S$.

But what happens when we string together these decisions into cycles? Now you are moving between elements of S, looking to your friend for help at each step as they move between elements of T. In this scenario, there are a few conditions that a lens $Sy^S \to Ty^T$ should satisfy to ensure that the associated delgation behaves well with respect to the movements of both you and your friend:

1. If your friend chooses to stay put, then you should stay put, too. This is reflected by the equation
$$\text{set}(s, \text{get}(s)) = s.$$
2. After your friend moves, and you move accordingly, you should delegate the decision at your new location to the decision at your friend's new location. This is reflected by the equation
$$\text{get}(\text{set}(s, t)) = t.$$
3. If your friend moves to t, then to t', the place where you end up should be where you would have ended up if your friend had moved directly to t' in the first place. This is reflected by the equation
$$\text{set}(\text{set}(s, t), t') = \text{set}(s, t')$$

We will see these three conditions emerge from more general theory in **??**.

2.3.4 Translating between natural transformations and lenses

We now know that we can specify a morphism of polynomials $p \to q$ in two ways:

- in the language of functors, by specifying a natural transformation $p \to q$, i.e. for each $X \in \textbf{Set}$, a function $p(X) \to q(X)$ such that naturality squares commute; or
- in the language of arenas, by specifying a lens $p \to q$, i.e. a function $f_1 \colon p(1) \to q(1)$ and, for each $i \in p(1)$, a function $f_i^\sharp \colon q[f_1(i)] \to p[i]$.

But what is the relationship between these two formulations? If you told me a lens between arenas, and I told you a natural transformation between polynomial functors, how could we tell if we were talking about the same morphism or not? We want to be able to translate between these two languages.

Our Rosetta Stone turns out to be the proof of the Yoneda lemma. The lemma itself forms the crux of the proof of Proposition 2.53, that these two formulations of polynomial morphisms are equivalent; so unraveling this proof reveals the translation we seek.

Proposition 2.70. Let p and q be polynomial functors, and let $(f_1, f^\sharp)$ be a morphism between their associated arenas. Then the isomorphism in (2.54) sends $(f_1, f^\sharp)$ to the natural transformation $f \colon p \to q$ whose X-component $f_X \colon p(X) \to q(X)$ for each $X \in \mathbf{Set}$ sends every

$$(i, g) \in \sum_{i \in p(1)} X^{p[i]} \cong p(X),$$

with $i \in p(1)$ and $g \colon p[i] \to X$, to

$$(f_1(i), f_i^\sharp \mathbin{⨾} g) \in \sum_{j \in q(1)} X^{q[j]} \cong q(X).$$

Proof. As an element of the product over I on the right hand side of (2.54), the pair $(f_1, f^\sharp)$ can equivalently be thought of as multiple pairs $((f_1(i), f_i^\sharp))_{i \in I}$. Fixing $i \in I$, the pair $(f_1(i), f_i^\sharp)$ is an element of

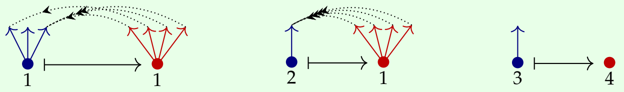
$$\sum_{j \in q(1)} p[i]^{q[j]} = q(p[i])$$

(so $f_1(i) \in q(1)$ and $f_i^\sharp \colon q[f_1(i)] \to p[i]$). By the Yoneda lemma (Lemma 2.10), we have an isomorphism $q(p[i]) \cong \mathbf{Poly}(y^{p[i]}, q)$, and by the proof of the Yoneda lemma, this isomorphism sends $(f_1(i), f_i^\sharp)$ to the natural transformation $f^i \colon y^{p[i]} \to q$ whose X-component is the function $f^i_X \colon X^{p[i]} \to q(X)$ given by sending $g \colon p[i] \to X$ to

$$q(g)(f_1(i), f_i^\sharp) = \left(\sum_{j \in q(1)} g^{q[j]}\right)(f_1(i), f_i^\sharp) = \left(f_1(i), g^{q[f_1(i)]}(f_i^\sharp)\right) = (f_1(i), f_i^\sharp \mathbin{⨾} g).$$

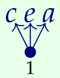
Taken together, the natural transformations $(f^i)_{i \in I}$ form an element of $\prod_{i \in I} \mathbf{Poly}(y^{p[i]}, q)$. Applying the universal property of coproducts, as in the proof of Proposition 2.53, we find that $(f^i)_{i \in I}$ corresponds to the natural transformation $f \colon p \to q$ we desire. □

Example 2.71. Let us return to the polynomials $p := y^3 + 2y$ and $q := y^4 + y^2 + 2$ from Example 2.56 and the morphism $f \colon p \to q$ depicted below:



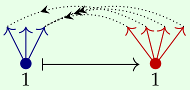
Fix a set $X := \{a, b, c, d, e\}$. When viewed as a natural transformation, the morphism f has as its X-component a function $f_X \colon p(X) \to q(X)$. In other words, for any element of $p(X)$, the morphism f should be able to give us an element of $q(X)$.

What does an element of $p(X)$ look like? Well, to specify such an element, we would need to choose a position i of p and a function $p[i] \to X$. We can depict this by selecting one of the corollas in the forest of p and labeling each leaf of that corolla with an element of X. For example, here we depict an element $(1, g)$ of $p(X)$, where $g\colon p[1] \to X$ is given by $1 \mapsto c, 2 \mapsto e$, and $3 \mapsto a$:

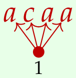


Similarly, an element of $q(X)$ can be drawn as a corolla in the forest of q, with each leaf labeled by an element of X. So what element of $q(X)$ is $f_X(1, g)$?

Proposition 2.70 tells us that $f_X(1, g)$ can be read off of the forest depiction of f at position 1:



To draw $f_X(1, g)$, we first draw the corolla in the forest of q corresponding to $f_1(1)$: the corolla on the right hand side above. Then we label each leaf of that corolla by following the arrow from that leaf (as given by $f_i^\sharp$) to a leaf of p at 1, and use the label there that is given by $(1, g)$. So $f_X(1, g)$ looks like



Proposition 2.70 lets us translate from lenses to natural transformations. The following corollary tells us how to go in the other direction. In particular, it justifies the notation f_1 for the on-positions function of f.

Corollary 2.72. Let p and q be polynomial functors, and let $f\colon p \to q$ be a natural transformation between them. Then the isomorphism in (2.54) sends f to the lens $(f_1, f^\sharp)$ for which $f_1\colon p(1) \to q(1)$ is the 1-component of f and, for each $i \in p(1)$, we have

$$(f_1(i), f_i^\sharp) = f_{p[i]}(i, \mathrm{id}_{p[i]}).$$

Proof. By Proposition 2.70, the 1-component of f is a function $p(1) \to q(1)$ sending every $i \in p(1)$ to $f_1(i) \in q(1)$, so the on-positions function f_1 is indeed equal to the 1-component of f. Also, the $p[i]$-component $f_{p[i]}\colon p(p[i]) \to q(p[i])$ sends every $(i, \mathrm{id}_{p[i]}) \in p(p[i])$, with $i \in p(1)$, to $(f_1(i), f_i^\sharp \fatsemi \mathrm{id}_{p[i]}) = (f_1(i), f_i^\sharp)$. □

2.3.5 Identity and composition of lenses

Thus far, we have seen how the category **Poly** of polynomial functors and natural transformations can just as easily be thought of as the category of arenas and lenses. But in order to actually discuss the latter category, we need to be able to give identity lenses and describe how these lenses compose. To do so, we can leverage our ability to translate back and forth between lenses and natural transformations.

For instance, given a polynomial p, the identity morphism of its associated arena should correspond to the identity natural transformation of p as a functor.

Exercise 2.73 (Identity lenses; solution here). Let p be a polynomial and let $\mathrm{id}_p \colon p \to p$ be its identity natural transformation, whose X-component $(\mathrm{id}_p)_X \colon p(X) \to p(X)$ for each $X \in \mathbf{Set}$ is the identity function on $p(X)$; that is, $(\mathrm{id}_p)_X = \mathrm{id}_{p(X)}$.

Use Corollary 2.72 to show that the lens $((\mathrm{id}_p)_1, (\mathrm{id}_p)^\sharp)$ associated to id_p is such that $(\mathrm{id}_p)_1 \colon p(1) \to p(1)$ and $(\mathrm{id}_p)^\sharp_i \colon p[(\mathrm{id}_p)_1(i)] \to p[i]$ for $i \in p(1)$ are all identity functions. ◊

Similarly, we should be able to deduce how two lenses compose by translating them to natural transformations, composing those, then translating back to lenses.

Exercise 2.74 (Composing lenses; solution here). Let p, q, and r be polynomials, let $f \colon p \to q$ and $g \colon q \to r$ be natural transformations, and let $h \coloneqq f \mathbin{⨟} g$ be their composite, whose X-component $h_X \colon p(X) \to r(X)$ for each $X \in \mathbf{Set}$ is the composite of the X-components of f and g; that is, $h_X = f_X \mathbin{⨟} g_X$.

Use Corollary 2.72 to show that the lens $(h_1, h^\sharp)$ associated to h satisfies $h_1 = f_1 \mathbin{⨟} g_1$ and $h^\sharp_i = g^\sharp_{f_1(i)} \mathbin{⨟} f^\sharp_i$ for all $i \in p(1)$. ◊

Example 2.75 (Commutative diagrams in **Poly**). The above exercise tells us how to interpret commutative diagrams in **Poly** as commutative diagrams in the more familiar setting of **Set**. Given polynomials p, q, r and natural transformations $f \colon p \to q, g \colon q \to r$, and $h \colon p \to r$, the diagram

$$\begin{tikzcd}
p \arrow[r, "f"] \arrow[dr, "h"'] & q \arrow[d, "g"] \\
& r
\end{tikzcd}$$

commutes in **Poly** if and only if the forwards on-positions diagram

$$\begin{tikzcd}
p(1) \arrow[r, "f_1"] \arrow[dr, "h_1"'] & q(1) \arrow[d, "g_1"] \\
& r(1)
\end{tikzcd}$$

commutes in **Set** and, for each $i \in p(1)$, the backwards on-directions diagram

$$\begin{tikzcd}
p[i] & q[f_1(i)] \arrow[l, "f^\sharp_i"'] \\
& r[h_1(i)] \arrow[u, "g^\sharp_{f_1(i)}"'] \arrow[ul, "h^\sharp_i"]
\end{tikzcd}$$

commutes in **Set**. We can use this fact to determine whether a given diagram in **Poly** commutes.

Exercise 2.76 (Solution here). Verify that, for $p, q \in \mathbf{Poly}$, the polynomial $p + q$ given by the binary sum of p and q satisfies the universal property of the coproduct of p and q. That is, provide morphisms $\iota \colon p \to p + q$ and $\kappa \colon q \to p + q$, then show that for any other polynomial r with morphisms $f \colon p \to r$ and $g \colon q \to r$, there exists a unique morphism $h \colon p + q \to r$—shown dashed—making the following diagram commute:

$$\begin{tikzcd}
p \arrow[r, "\iota"] \arrow[dr, "f"'] & p + q \arrow[d, dashed, "h"] & q \arrow[l, "\kappa"'] \arrow[dl, "g"] \\
& r &
\end{tikzcd} \tag{2.77}$$

Hint: Use Example 2.75 to determine whether a diagram commutes. ◊

Exercise 2.78 (A functor **Top** → **Poly**; solution here). This exercise is for those who know what topological spaces and continuous maps are. It will not be used again in this book.

1. Suppose that X is a topological space. Organize its points and their neighborhoods into a polynomial p_X.
2. Give a formula by which any continuous map $X \to Y$ induces a map of polynomials $p_X \to p_Y$.
3. Show that your formula defines a functor.
4. Is it full? Faithful? ◊

2.4 Prepare for dynamics

As beautiful as the category **Poly** is—and to be clear we have not really begun to say what is so special about it—discussing its virtues is not our goal. We want to use it!

Polynomial functors are a setting in which to speak about dynamics, decisions, and data. We want the reader to understand this deeply as they go through the mathematics. So in order to make the story a bit more seamless, we discuss a few relevant aspects of **Poly** that we can use immediately.

2.4.1 The categorical product

The category **Poly** has limits and colimits, is cartesian closed, has epi-mono factorizations, etc., etc. However, in order to tell a good story of dynamics, we only need products right now. These will be useful for letting many different interfaces control the same internal dynamics.

Proposition 2.79. The category **Poly** has arbitrary products, coinciding with products in $[\mathbf{Set}, \mathbf{Set}]$ given by the operation $\prod_{i \in I}$.

Proof. The proof is very similar to that of Proposition 2.50.

By Example 2.34, the category [**Set**, **Set**] has arbitrary products given by $\prod_{i\in I}$. The full subcategory inclusion **Poly** → [**Set**, **Set**] reflects these products. It remains to show that **Poly** is closed under the operation $\prod_{i\in I}$.

By Proposition 2.35, [**Set**, **Set**] is completely distributive. Hence, given polynomials $(p_i)_{i\in I}$, we can use (2.28) to write their product in [**Set**, **Set**] as

$$\prod_{i\in I} p_i \cong \prod_{i\in I}\sum_{j\in p_i(1)} y^{p_i[j]} \cong \sum_{\bar{j}\in\prod_{i\in I} p_i(1)}\prod_{i\in I} y^{p_i[\bar{j}(i)]} \cong \sum_{\bar{j}\in\prod_{i\in I} p_i(1)} y^{\sum_{i\in I} p_i[\bar{j}(i)]}, \tag{2.80}$$

which, as a coproduct of representables, is in **Poly**. □

Corollary 2.81. The category **Poly** is completely distributive.

Proof. This is a direct consequence of the fact that **Poly** has arbitrary (co)products coinciding with (co)products in [**Set**, **Set**] (Propositions 2.50 and 2.79) and the fact that [**Set**, **Set**] itself is completely distributive (Proposition 2.35). □

The result above will allow us to apply Eq. (2.28), or sometimes specifically Eq. (2.30), to push $\prod$'s past $\sum$'s of polynomials whenever we so desire.

Exercise 2.82 (Solution here).

Use (2.54) to verify that

$$\mathbf{Poly}\left(q, \prod_{i\in I} p_i\right) \cong \prod_{i\in I} \mathbf{Poly}(q, p_i)$$

for all polynomials $(p_i)_{i\in I}$ and q, as one would expect from the universal property of products. ◊

Exercise 2.83 (Solution here). Let $p_1 := y+1$, $p_2 := y+2$, and $p_3 := y^2$. What is $\prod_{i\in 3} p_i$ according to (2.80)? Is the answer what you would expect? ◊

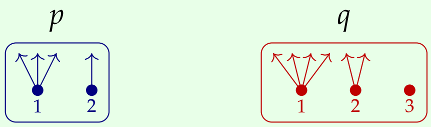
It follows from (2.80) that the terminal object of **Poly** is 1, and that binary products are given by

$$p \times q \cong \sum_{i\in p(1)}\sum_{j\in q(1)} y^{p[i]+q[j]}. \tag{2.84}$$

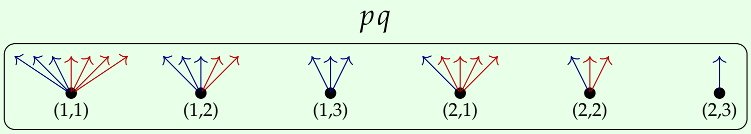
We will sometimes write pq rather than $p \times q$:

$$pq := p \times q \tag{Notation}$$

Example 2.85. We can draw the product of two polynomials in terms of their associated forests. Let $p := y^3 + y$ and $q := y^4 + y^2 + 1$.



Then $pq \cong y^7 + 2y^5 + 2y^3 + y$. As arenas, we take all pairs of positions, and for each pair we take the disjoint union of the directions.



In practice, we can multiply polynomial functors the same way we would multiply two polynomials in high school algebra.

Exercise 2.86 (Solution here).

1. Show that for sets A_1, B_1, A_2, B_2, we have

$$A_1 y^{B_1} \times A_2 y^{B_2} \cong A_1 A_2 y^{B_1 + B_2}.$$

2. Show that for sets $(A_i)_{i \in I}$ and $(B_j)_{j \in J}$, we have

$$\left(\sum_{i \in I} A_i y^{B_i} \right) \times \left(\sum_{j \in J} A_j y^{B_j} \right) \cong \sum_{i \in I} \sum_{j \in J} A_i A_j y^{B_i + B_j}.$$

◊

As lenses, the canonical projections $\pi \colon pq \to p$ and $\phi \colon pq \to q$ behave as you might expect: on positions, they are the projections from $(pq)(1) \cong p(1) \times q(1)$ to $p(1)$ and $q(1)$, respectively; on directions, they are the inclusions $p[i] \to p[i] + q[j]$ and $q[j] \to p[i] + q[j]$ for each position (i, j) of pq.

Exercise 2.87 (Solution here). Verify that, for $p, q \in$ **Poly**, the polynomial pq given by (2.84) along with the maps $\pi \colon pq \to p$ and $\phi \colon pq \to q$ described above satisfy the universal property of the product of p and q. ◊

2.4.2 The parallel product

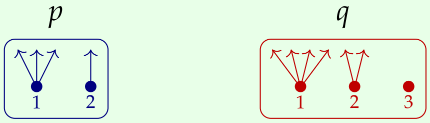
There is a closely related monoidal structure on **Poly** that will be useful for putting dynamical systems in parallel.

Definition 2.88. Let p and q be polynomials. When they are expressed in standard notation, their *parallel product*, denoted $p \otimes q$, is given by the formula:

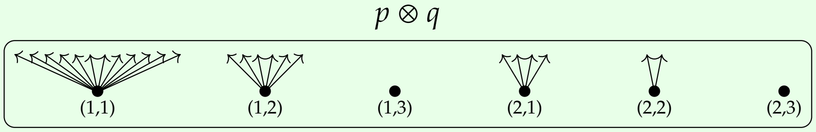
$$p \otimes q \cong \sum_{i \in p(1)} \sum_{j \in q(1)} y^{p[i] \times q[j]}. \tag{2.89}$$

One should compare this with the formula for the product of polynomials shown in (2.84). The difference is that the parallel product multiplies exponents where the categorical product adds them.

Example 2.90. We can draw the parallel product of two polynomials in terms of their associated forests. Let $p := y^3 + y$ and $q := y^4 + y^2 + 1$.



Then $p \otimes q \cong y^{12} + y^6 + y^4 + y^2 + 2$. As arenas, we take all pairs of positions, and for each pair we take the product of the directions.



Exercise 2.91 (Solution here).

1. Show that for sets A_1, B_1, A_2, B_2, we have
$$A_1 y^{B_1} \otimes A_2 y^{B_2} \cong A_1 A_2 y^{B_1 B_2}.$$
2. Show that for sets $(A_i)_{i \in I}$ and $(B_j)_{j \in J}$, we have
$$\left(\sum_{i \in I} A_i y^{B_i}\right) \otimes \left(\sum_{j \in J} A_j y^{B_j}\right) \cong \sum_{i \in I} \sum_{j \in J} A_i A_j y^{B_i B_j}.$$

◊

Exercise 2.92 (Solution here). Let $p := y^2 + y$ and $q := 2y^4$.

1. Draw p and q as corolla forests.
2. Draw $pq = p \times q$ as a corolla forest.
3. Draw $p \otimes q$ as a corolla forest. ◊

Exercise 2.93 (Solution here). Consider the polynomials $p \coloneqq 2y^2 + 3y$ and $q \coloneqq y^4 + 3y^3$.

1. What is $p \times q$?
2. What is $p \otimes q$?
3. What is the product of the following purely formal expression we'll see *only this once!*:
$$(2 \cdot 2^y + 3 \cdot 1^y) \cdot (1 \cdot 4^y + 3 \cdot 3^y)$$
The factors of the above product are called Dirichlet series.
4. Describe the connection between the last two parts. (An alternative name we give for the parallel product $\otimes$ is the *Dirichlet product*.) ◊

Exercise 2.94 (Solution here). What is $(3y^5 + 6y^2) \otimes 4$? Hint: $4 = 4y^0$. ◊

Exercise 2.95 (Solution here). Let $p, q, r \in \mathbf{Poly}$ be any polynomials.

1. Show that there is an isomorphism $p \otimes y \cong p$.
2. Show that there is an isomorphism $(p \otimes q) \otimes r \cong p \otimes (q \otimes r)$.
3. Show that there is an isomorphism $p \otimes q \cong q \otimes p$. ◊

In Exercise 2.95, we have gone most of the way to proving that $(\mathbf{Poly}, y, \otimes)$ is a symmetric monoidal category.

Proposition 2.96. The category **Poly** has a symmetric monoidal structure $(y, \otimes)$ where $\otimes$ is the parallel product from Definition 2.88.

Sketch of proof. Given lenses $f\colon p \to p'$ and $g\colon q \to q'$, we need to give a map $(f \otimes g)\colon (p \otimes q) \to (p' \otimes q')$. On positions, define

$$(f \otimes g)_1(i, j) \coloneqq \big(f_1(i), g_1(j)\big)$$

On directions at $(i, j) \in p(1) \times q(1)$, define

$$(f \otimes g)^\sharp_{(i,j)}(d, e) \coloneqq \big(f^\sharp_i(d), g^\sharp_j(e)\big).$$

Then Exercise 2.95 gives us the unitors, associator, and braiding. We have not proven the functoriality of $\otimes$, the naturality of the isomorphisms from Exercise 2.95, or all the coherences between these isomorphisms, but we ask the reader to take them on trust or to check them for themselves. Alternatively, we may invoke the Day convolution to obtain the monoidal structure $(y, \otimes)$ directly (see Proposition 2.101). □

Exercise 2.97 (Solution here).

1. If $p = A$ and $q = B$ are constant polynomials, what is $p \otimes q$?
2. If $p = A$ is constant and q is arbitrary, what can you say about $p \otimes q$?
3. If $p = Ay$ and $q = By$ are linear polynomials, what is $p \otimes q$?
4. For arbitrary $p, q \in \mathbf{Poly}$, what is the relationship between the sets $(p \otimes q)(1)$ and $p(1) \times q(1)$? ◊

Exercise 2.98 (Solution here). Which of the following classes of polynomials are closed under $\otimes$? Note also whether they contain y.

1. The set $\{Ay^0 \mid A \in \mathbf{Set}\}$ of constant polynomials.
2. The set $\{Ay \mid A \in \mathbf{Set}\}$ of linear polynomials.
3. The set $\{Ay + B \mid A, B \in \mathbf{Set}\}$ of affine polynomials.
4. The set $\{Ay^2 + By + C \mid A, B, C \in \mathbf{Set}\}$ of quadradic polynomials.
5. The set $\{Ay^B \mid A, B \in \mathbf{Set}\}$ of monomials.
6. The set $\{Sy^S \mid S \in \mathbf{Set}\}$ of systematic polynomials.
7. The set $\{p \in \mathbf{Poly} \mid p(1) \text{ is finite}\}$. ◊

Exercise 2.99 (Solution here). What is the smallest class of polynomials that's closed under $\otimes$ and contains y? ◊

Exercise 2.100 (Solution here). Show that for any $p_1, p_2, q \in \mathbf{Poly}$ there is an isomorphism

$$(p_1 + p_2) \otimes q \cong (p_1 \otimes q) + (p_2 \otimes q).$$ ◊

Proposition 2.101. For any monoidal structure $(I, \star)$ on **Set**, there is a corresponding monoidal structure $(y^I, \odot)$ on **Poly**, where $\odot$ is the Day convolution. Moreover, $\odot$ distributes over coproducts.

In the case of $(0, +)$ and $(1, \times)$, this procedure returns the $(1, \times)$ and $(y, \otimes)$ monoidal structures respectively.

Proof. Any monoidal structure $(I, \odot)$ on **Set** induces a monoidal structure on $[\mathbf{Set}, \mathbf{Set}]$ with the Day convolution $\odot$ as the tensor product and y^I as the unit. To prove that this monoidal structure restricts to **Poly**, it suffices to show that **Poly** is closed under the Day convolution.

Given polynomials $p \coloneqq \sum_{i \in p(1)} y^{p[i]}$ and $q \coloneqq \sum_{j \in q(1)} y^{q[j]}$, their Day convolution is given by the coend

$$p \odot q \cong \int^{(A,B) \in \mathbf{Set}^2} y^{A \star B} \times p(A) \times q(B). \tag{2.102}$$

We can rewrite the product $p(A) \times q(B)$ as

$$p(A) \times q(B) \cong \left(\sum_{i \in p(1)} A^{p[i]}\right) \times \left(\sum_{j \in q(1)} B^{q[i]}\right) \cong \sum_{(i,j) \in p(1) \times q(1)} A^{p[i]} \times B^{q[i]}$$

So because products distribute over coproducts in **Set** and coends commute with coproducts, we can rewrite (2.102) as

$$p \odot q \cong \sum_{(i,j) \in p(1) \times q(1)} \int^{(A,B) \in \mathbf{Set}^2} y^{A \star B} \times A^{p[i]} \times B^{q[i]},$$

which, by the co-Yoneda lemma, can be rewritten as

$$p \odot q \cong \sum_{(i,j) \in p(1) \times q(1)} y^{p[i] \star q[j]} \tag{2.103}$$

in **Poly**. That the Day convolution distributes over coproducts also follows from the fact that products distribute over coproducts in **Set** and coends commute with coproducts; or, alternatively, directly from (2.103).

We observe that (2.103) gives $(y^I, \odot) = (1, \times)$ when $(I, \star) = (0, +)$ and $(y^I, \odot) = (y, \otimes)$ when $(I, \star) = (1, \times)$. □

Exercise 2.104 (Solution here).

1. Show that the operation $(A, B) \mapsto A + AB + B$ on **Set** is associative.
2. Show that 0 is unital for the above operation.
3. Let $(1, \odot)$ denote the corresponding monoidal structure on **Poly**. Compute the monoidal product $(y^3 + y) \odot (2y^2 + 2)$. ◊

Remark 2.105. Monoids in **Poly** with respect to the parallel product $\otimes$ are particularly interesting—they have a kind of collective semantics, letting agents aggregate their contributions and distribute returns on those contributions in a coherent way. We leave discussion of them to future work, so as not to distract us from our main story.

2.5 Summary and further reading

Thanks to Joachim Kock for telling us about the derivative $\dot{p}$ of a polynomial and the relationship between $\dot{p}(1)$ and the total number of leaves of p.

2.6 Exercise solutions

Solution to Exercise 2.4.

Our goal is to say whether various functors are representable and, for those that are, what the representing set is.

1. The identity functor $X \mapsto X$ is represented by $S = 1$: a function $1 \to X$ is just an element of X, so $\mathbf{Set}(1, X) \cong X$. Alternatively, note that $X^1 \cong X$.
2. The constant functor $X \mapsto 2$ is not representable: the functor sends 1 to 2, but $1^S \cong 1 \ncong 2$ for any set S.
3. The constant functor $X \mapsto 1$ is representable by $S = 0$: there is exactly one function $0 \to X$, so $\mathbf{Set}(0, X) \cong 1$. Alternatively, note that $X^0 \cong 1$.
4. The constant functor $X \mapsto 0$ is not representable, for the same reason as in #2.
5. The functor $y^{\mathbb{N}}$ that sends $X \mapsto X^{\mathbb{N}}$ is represented by $S = \mathbb{N}$, by definition. It sends each function $h \colon X \to Y$ to the function $h^{\mathbb{N}} \colon X^{\mathbb{N}} \to Y^{\mathbb{N}}$ that sends each $g \colon \mathbb{N} \to X$ to $g \fatsemi h \colon \mathbb{N} \to Y$.
6. No $\mathbf{Set} \to \mathbf{Set}$ functor $X \mapsto 2^X$ is representable, for the same reason as in #2. (There *is*, however, a functor $\mathbf{Set}^{\mathrm{op}} \to \mathbf{Set}$ sending $X \mapsto 2^X$ that is understood to be representable in a more general sense.)

Solution to Exercise 2.6.

To show that

$$\begin{tikzcd}
X^S \arrow[r, "h^S"] \arrow[d, "X^f"'] \arrow[dr, phantom, "?"] & Y^S \arrow[d, "Y^f"] \\
X^R \arrow[r, "h^R"'] & Y^R
\end{tikzcd}$$

commutes, we note that by Proposition 2.5, both vertical maps compose functions from S with $f \colon R \to S$ from the left, and by Definition 2.1, both horizontal maps compose functions to X with $h \colon X \to Y$ on the right. So by the associativity of composition, the diagram commutes.

Solution to Exercise 2.7.

In each case, given $f \colon R \to S$, we can find the X-component $X^f \colon X^S \to X^R$ of the natural transformation $y^f \colon y^S \to y^R$ by applying Proposition 2.5, which says that X^f sends each $g \colon S \to X$ to $f \fatsemi g \colon R \to X$.

1. If $R = 5$, $S = 5$, and $f = \mathrm{id}$, then X^f is the identity function on X^5.
2. If $R = 2$, $S = 1$, and f is the unique function, then X^f sends each $g \in X$ (i.e. function $g \colon 1 \to X$) to the function that maps both elements of 2 to g. We can think of X^f as the diagonal $X \to X \times X$.
3. If $R = 1$, $S = 2$, and $f(1) = 1$, then X^f sends each $g \colon 2 \to X$ to $g(1)$, viewed as a function $1 \to X$. We can think of X^f as the left projection $X \times X \to X$.
4. If $R = 1$, $S = 2$, and $f(1) = 2$, then X^f sends each $g \colon 2 \to X$ to $g(2)$, viewed as a function $1 \to X$. We can think of X^f as the right projection $X \times X \to X$.
5. If $R = 0$, $S = 5$, and f is the unique function, then X^f is the unique function $X^5 \to X^0 \cong 1$.
6. If $R = \mathbb{N}$, $S = \mathbb{N}$, and $f(n) = n + 1$, then X^f sends each $g \colon \mathbb{N} \to X$ to the function $h \colon \mathbb{N} \to X$ satisfying $h(n) = g(n + 1)$ for all $n \in \mathbb{N}$. We can think of X^f as removing the first term of an infinite sequence of elements of X.

Solution to Exercise 2.8.

1. The fact that $y^{\mathrm{id}_S} \colon y^S \to y^S$ is the identity is just a generalization of Exercise 2.7 #1. For any set X, the X-component $X^{\mathrm{id}_S} \colon X^S \to X^S$ of y^{id_S} sends each $h \colon S \to X$ to $\mathrm{id}_S \fatsemi h = h$, so X^{id_S} is the identity on X^S. Hence y^{id_S} is the identity on y^S.
2. Fix $f \colon R \to S$ and $g \colon S \to T$; we wish to show that $y^g \fatsemi y^f = y^{f \fatsemi g}$. It suffices to show componentwise that $X^g \fatsemi X^f = X^{f \fatsemi g}$ for every set X. Indeed, X^g sends each $h \colon T \to X$ to $g \fatsemi h$; then X^f sends $g \fatsemi h$ to $f \fatsemi g \fatsemi h = X^{f \fatsemi g}(h)$.

Solution to Exercise 2.12.

1. Given $a \in F(S)$, naturality of the maps $X^S \to F(X)$ that send $g\colon S \to X$ to $F(g)(a)$ amounts to the commutativity of

$$\begin{tikzcd}
X^S \arrow[r, "h^S"] \arrow[d, "F(-)(a)"'] & Y^S \arrow[d, "F(-)(a)"] \\
F(X) \arrow[r, "F(h)"'] & F(Y)
\end{tikzcd}$$

for all $h\colon X \to Y$. The top map h^S sends any $g\colon X \to S$ to $g \fatsemi h$ (Definition 2.1), which is then sent to $F(g \fatsemi h)(a)$ by the right map. Meanwhile, the left map sends g to $F(g)(a)$, which is then sent to $F(h)(F(g)(a))$ by the bottom map. So by the functoriality of F, the square commutes.

2. We seek to show that the two maps from the proof sketch of Lemma 2.10 are mutually inverse. First, we show that for any natural transformation $m\colon y^S \to F$, we have that $m^{m_S(\mathrm{id}_S)} = m$. Given a set X, the X-component of $m^{m_S(\mathrm{id}_S)}$ sends each $g\colon S \to X$ to $F(g)(m_S(\mathrm{id}_S))$; it suffices to show that this is also where the X-component of m sends g. Indeed, by the naturality of m, the square

$$\begin{tikzcd}
S^S \arrow[r, "g^S"] \arrow[d, "m_S"'] & X^S \arrow[d, "m_X"] \\
F(S) \arrow[r, "F(g)"'] & F(X)
\end{tikzcd}$$

commutes, so in particular

$$F(g)(m_S(\mathrm{id}_S)) = m_X(g^S(\mathrm{id}_S)) = m_X(\mathrm{id}_S \fatsemi g) = m_X(g). \tag{2.106}$$

In the other direction, we show that for any $a \in F(S)$, we have $m^a_S(\mathrm{id}_S) = a$: by construction, $m^a_S\colon S^S \to F(S)$ sends id_S to $F(\mathrm{id}_S)(a) = a$.

3. Given functors $F, G\colon [\mathbf{Set}, \mathbf{Set}]$ and a natural transformation $\alpha\colon F \to G$, we wish to show that the naturality square

$$\begin{tikzcd}
\mathsf{Nat}(y^S, F) \arrow[r, "\sim"] \arrow[d, "- \fatsemi \alpha"'] & F(S) \arrow[d, "\alpha_S"] \\
\mathsf{Nat}(y^S, G) \arrow[r, "\sim"] & G(S)
\end{tikzcd}$$

commutes. The top map sends any $m\colon y^S \to F$ to $m_S(\mathrm{id}_S)$, which in turn is sent by the right map to $\alpha_S(m_S(\mathrm{id}_S)) = (m \fatsemi \alpha)_S(\mathrm{id}_S)$. This is also where the bottom map sends $m \fatsemi \alpha$, so the square commutes.

4. Given a function $g\colon S \to X$, we wish to show that the naturality square on the left side of the diagram

$$\begin{tikzcd}
\mathsf{Nat}(y^S, F) \arrow[r, "\sim"] \arrow[d, "y^g \fatsemi -"'] & F(S) \arrow[d, "F(f)"] \\
\mathsf{Nat}(y^X, F) \arrow[r, "\sim"] & F(X)
\end{tikzcd}$$

commutes. The left map sends any $m\colon y^S \to F$ to $y^g \fatsemi m$, which is sent by the bottom map to $(y^g \fatsemi m)_X(\mathrm{id}_X) = m_X(X^g(\mathrm{id}_X)) = m_X(f \fatsemi \mathrm{id}_X) = m_X(g)$. Meanwhile, the top map sends m to $m_S(\mathrm{id}_S)$, which is sent by the right map to $F(g)(m_S(\mathrm{id}_S))$. So the square commutes by (2.106).

5. To show that $\mathsf{Nat}(y^S, y^T) \cong S^T$, just take $F = y^T$ in Lemma 2.10.

Solution to Exercise 2.16.

We are given a set I and a dependent set $(X_i)_{i \in I}$ for which $X_i \coloneqq 1$ for every $i \in I$.

1. To show that $I \cong \sum_{i \in I} 1$, we note that $x \in 1 = \{1\}$ if and only if $x = 1$, so $\sum_{i \in I} 1 = \{(i, 1) \mid i \in I\}$. Then function $I \to \sum_{i \in I} 1$ that sends each $i \in I$ to $(i, 1)$ is clearly an isomorphism.
2. To show that $1 \cong \prod_{i \in I} 1$, it suffices to demonstrate that there is a unique dependent function $f\colon (i \in I) \to 1$. As $1 = \{1\}$, such a function f must always send $i \in I$ to 1. This uniquely characterizes f, so there is only one such dependent function.

Now $I := \varnothing$ and $X\colon \varnothing \to \mathbf{Set}$ is the unique empty collection of sets.

3. Yes: since I is empty, there are no $i \in I$. So it is true that $X_i = 1$ holds whenever $i \in I$ holds, because $i \in I$ never holds. We say that this sort of statement is "vacuously true."
4. As $I = \varnothing = \mathsf{0}$, we have $\mathsf{0} = I \cong \sum_{i \in I} \mathsf{1} = \sum_{i \in \varnothing} X_i$, where the middle isomorphism follows from #1 and the last equation follows from #3.
5. As $I = \varnothing = \mathsf{0}$, we have $\mathsf{1} \cong \prod_{i \in I} \mathsf{1} = \prod_{i \in \varnothing} X_i$, where the isomorphism on the left follows from #2 and the equation on the right follows from #3.

Solution to Exercise 2.18.

We have a dependent set $X\colon I \to \mathbf{Set}$ and a projection function $\pi_1\colon \sum_{i \in I} X_i \to I$ defined by $\pi_1(i, x) = i$.

1. The second projection $\pi_2(i, x) = x$ sends each pair $p = (i, x) \in \sum_{i \in I} X_i$ to x, an element of X_i. Note that we can write i in terms of p as $\pi_1(p)$. This allows us to write the signature of π_2 as $\pi_2\colon (p \in \sum_{i \in I} X_i) \to X_{\pi_1(p)}$.
2. Let $S := \{s\colon I \to \sum_{i \in I} X_i \mid s \mathbin{⨾} \pi_1 = \mathrm{id}_I\}$ be the set of sections of π_1. To show that $\prod_{i \in I} X_i \cong S$, we will exhibit maps in either direction and show that they are mutually inverse. For each $f\colon (i \in I) \to X_i$ in $\prod_{i \in I} X_i$, we have that $f(i) \in X_i$ for all $i \in I$, so we can define a function $s_f\colon I \to \sum_{i \in I} X_i$ that sends each $i \in I$ to $(i, f(i))$. Then $\pi_1(s_f(i)) = \pi_1(i, f(i)) = i$, so s_f is indeed a section of π_1. Hence $f \mapsto s_f$ is a map $\prod_{i \in I} X_i \to S$.
   In the other direction, for each section $s\colon I \to \sum_{i \in I} X_i$ we have $\pi_1(s(i)) = i$ for all $i \in I$, so we can write $s(i)$ as an ordered pair $(i, \pi_2(s(i)))$ with $\pi_2(s(i)) \in X_i$. It follows that we can define a function $f_s\colon (i \in I) \to X_i$ that sends each $i \in I$ to $\pi_2(s(i))$. Then $s \mapsto f_s$ is a map $S \to \prod_{i \in I} X_i$. By construction $s_{f_s}(i) = (i, f_s(i)) = (\pi_1(s(i)), \pi_2(s(i))) = s(i)$ and $f_{s_f}(i) = \pi_2(s_f(i)) = \pi_2(i, f(i)) = f(i)$, so these maps are mutually inverse.

Solution to Exercise 2.21.

We are given the set

$$B := \prod_{i \in I} \sum_{j \in J(i)} \prod_{k \in K(i,j)} X(i, j, k).$$

1. Here are the instructions for choosing an element of B as a nested list.

```
To choose an element of $B$:
  1. for each element $i \in I$:
    1.1. choose an element $j \in J(i)$;
    1.2. for each element $k \in K(i, j)$:
      1.2.1. choose an element of $X(i, j, k)$.
```

2. Given $I := \{1, 2\}$, $J(1) := \{j\}$, $J(2) := \{j, j'\}$, $K(1, j) := \{k_1, k_2\}$, $K(2, j) := \{k_1\}$, $K(2, j') := \{k'\}$, and $X(i, j, k) = \{x, y\}$ for all i, j, k, our goal is to count the number of elements in B. To compute the cardinality of B, we can use the fact that the cardinality of a sum (resp. product) is the sum (resp. product) of the cardinalities. So

$$\begin{aligned} |B| &= \prod_{i \in I} \sum_{j \in J(i)} \prod_{k \in K(i,j)} |X(i, j, k)| \\ &= \prod_{i \in \{1,2\}} \sum_{j \in J(i)} \prod_{k \in K(i,j)} 2 \\ &= \left(\sum_{j \in J(1)} 2^{|K(1,j)|} \right) \left(\sum_{j \in J(2)} 2^{|K(2,j)|} \right) \\ &= \left(2^2\right) \left(2^1 + 2^1\right) = 16. \end{aligned}$$

3. Here are three of the elements of B (you may have written down others):
   - $(1 \mapsto (j, k_1 \mapsto x, k_2 \mapsto y), 2 \mapsto (j', k' \mapsto x))$
   - $(1 \mapsto (j, k_1 \mapsto y, k_2 \mapsto y), 2 \mapsto (j, k_1 \mapsto y))$

- $(1 \mapsto (j, k_1 \mapsto y, k_2 \mapsto x), 2 \mapsto (j', k' \mapsto y))$

Solution to Exercise 2.31.

Given objects X, Y, Z in a completely distributive category $\mathcal{C}$, we wish to show that $X(Y+Z) \cong XY+XZ$ using (2.28). On the left hand side, we are taking a 2-fold product: a single object times a 2-fold sum. So we should let $I \coloneqq \mathsf{2}$ and let $J(1) \coloneqq \mathsf{1}$, with $X(1,1) \coloneqq X$, and $J(2) \coloneqq \mathsf{2}$, with $X(2,1) \coloneqq Y$ and $X(2,2) \coloneqq Z$. Then (2.28) says that

$$X(Y+Z) \cong \sum_{\bar{j}\in\prod_{i\in\mathsf{2}} J(i)} \prod_{i\in\mathsf{2}} X(i, \bar{j}(i)) \cong \sum_{\bar{j}\in\prod_{i\in\mathsf{2}} J(i)} X(1, \bar{j}(1)) \times X(2, \bar{j}(2)).$$

The set $\prod_{i\in\mathsf{2}} J(i)$ contains two functions: $(1 \mapsto 1, 2 \mapsto 1)$ and $(1 \mapsto 1, 2 \mapsto 2)$. So we can rewrite the right hand side as

$$X(1,1) \times X(2,1) + X(1,1) \times X(2,2) \cong XY + XZ,$$

as desired.

Solution to Exercise 2.32.

Our goal is to rewrite each expression so that every $\sum$ appears before every $\prod$.

1. By applying (2.28), we can rewrite

$$\sum_{i\in I} \prod_{j\in J(i)} \sum_{k\in K(i,j)} X(i,j,k)$$

as

$$\sum_{i\in I} \sum_{\bar{k}\in\prod_{j\in J} K(i,j)} \prod_{j\in J(i)} X(i,j,\bar{k}(j)).$$

2. By applying (2.28), we can rewrite

$$\prod_{i\in I} \sum_{j\in J(i)} \prod_{k\in K(i,j)} X(i,j,k)$$

as

$$\sum_{\bar{j}\in\prod_{i\in I} J(i)} \prod_{i\in I} X(i, \bar{j}(i)) \prod_{k\in K(i,\bar{j}(i))} X(i, \bar{j}(i), k).$$

3. By applying (2.28), we can rewrite

$$\prod_{i\in I} \prod_{j\in J(i)} \sum_{k\in K(i,j)} X(i,j,k)$$

once as

$$\prod_{i\in I} \sum_{\bar{k}\in\prod_{j\in J} K(i,j)} \prod_{j\in J(i)} X(i,j,\bar{k}(j))$$

and then again as

$$\sum_{\bar{\bar{k}}\in\prod_{i\in I}\prod_{j\in J} K(i,j)} \prod_{i\in I} \prod_{j\in J(i)} X(i,j,\bar{\bar{k}}(i,j)).$$

Solution to Exercise 2.36.

1. It suffices to show that for all $X \in \mathbf{Set}$, there is an isomorphism

$$\sum_{i\in I} F(X) \cong (IF)(X)$$

natural in X. The left hand side is naturally isomorphic to the set $\{(i,s) \mid i \in I \text{ and } s \in F(X)\} \cong I \times F(X)$, while the right hand side is also naturally isomorphic to the set $I(X) \times F(X) \cong I \times F(X)$. (Alternatively, since $[\mathbf{Set}, \mathbf{Set}]$ is completely distributive by Proposition 2.35, the result also follows from (2.28).)

2. It doesn't matter: $2y$ can denote either one. Indeed, taking $I \coloneqq 2$, the above says that there is an isomorphism $2 \times y \cong y + y$.

Solution to Exercise 2.37.

1. It suffices to show that for all $X \in \mathbf{Set}$, there is an isomorphism
$$\prod_{i \in I} y(X) \cong y^I(X).$$
natural in X. We have that $y(X) \cong X$ and that $y^I(X) \cong X^I$. So both sides are naturally isomorphic to the set of functions $I \to X$.
2. It suffices to show that for all $X \in \mathbf{Set}$, there is an isomorphism
$$\prod_{i \in I} y^A(X) \cong y^{IA}(X).$$
natural in X. We have that $y^A(X) \cong X^A$ and that $y^{IA}(X) \cong X^{IA}$. By currying, both sides are naturally isomorphic to the set of functions $I \times A \to X$.

Solution to Exercise 2.42.

Our goal is to write out the elements of $p(X)$ for each polynomial p and set X that we are given.

1. If $p \coloneqq y^3$ and $X \coloneqq \{4, 9\}$, then let $I \coloneqq 1$ and $p[1] \coloneqq 3$ so that $p \cong \sum_{i \in I} y^{p[i]}$. So
$$p(X) \cong \{(1,4,4,4),(1,4,4,9),(1,4,9,4),(1,4,9,9),(1,9,4,4),(1,9,4,9),(1,9,9,4),(1,9,9,9)\}.$$
2. If $p \coloneqq 3y^2 + 1$ and $X \coloneqq \{a\}$, then let $I \coloneqq 4$, $p[1] \coloneqq p[2] \coloneqq p[3] \coloneqq 2$, and $p[4] \coloneqq 1$ so that $p \cong \sum_{i \in I} y^{p[i]}$. So $p(X) \cong \{(1,a,a),(2,a,a),(3,a,a),(4)\}$.
3. If $p \coloneqq 0$ and $X \coloneqq \mathbb{N}$, then let $I \coloneqq 0$ so that $p \cong \sum_{i \in I} y^{p[i]}$. So $p(X) \cong 0$. Alternatively, we note that 0 is the constant functor that assigns 0 to every set.
4. If $p \coloneqq 4$ and $X \coloneqq \mathbb{N}$, then let $I \coloneqq 4$ and $p[i] \coloneqq 0$ for every $i \in I$. So $p(X) \cong \{(1),(2),(3),(4)\} \cong 4$.
5. If $p \coloneqq y$ and $X \coloneqq \mathbb{N}$, then let $I \coloneqq 1$ and $p[1] \coloneqq 1$ so that $p \cong \sum_{i \in I} y^{p[i]}$. So $p(X) \cong \{(1,n) \mid n \in \mathbb{N}\}$.

Solution to Exercise 2.45.

We consider $p(0)$ for arbitrary polynomials p. A representable functor y^S for some $S \in \mathbf{Set}$ sends $0 \mapsto 0$ if $S \neq 0$ (as there are then no functions $S \to 0$), but sends $0 \mapsto 1$ if $S = 0$ (as there is a unique function $0 \to 0$). So
$$p(0) \cong \sum_{i \in p(1)} (y^{p[i]})(0) \cong \sum_{\substack{i \in p(1),\\ p[i] \neq 0}} 0 + \sum_{\substack{i \in p(1),\\ p[i] = 0}} 1 \cong \{i \in p(1) \mid p[i] = 0\}.$$

In other words, $p(0)$ is the set of *constant* representable summands of p. For example, if $p \coloneqq y^3 + 3y^2 + 4$, then $p(0) = 4$. In the language of high school algebra, we might call $p(0)$ the "constant term" of p.

Solution to Exercise 2.48.

Given $p \coloneqq y^2 + y$, we seek the induced function $p(f)\colon p(1) \to p(2)$ for a function $f\colon 1 \to 2$ of our choice. We will choose $1 \mapsto 2$. We can evaluate
$$p(1) \cong \{(1,1,1),(2,1)\} \text{ and } p(2) \cong \{(1,1,1),(1,1,2),(1,2,1),(1,2,2),(2,1),(2,2)\}.$$

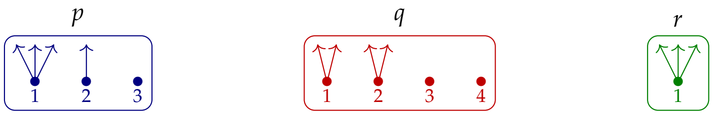
So $p(f)$ sends $(1,1,1) \mapsto (1,2,2)$ and $(2,1) \mapsto (2,2)$. (If we had instead picked $1 \mapsto 1$ as our function f, then $p(f)$ would send $(1,1,1) \mapsto (1,1,1)$ and $(2,1) \mapsto (2,1)$.)

Solution to Exercise 2.52.

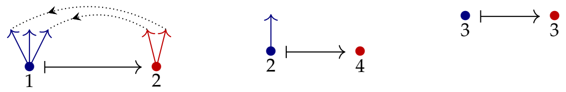
As $\mathbf{Poly}(y^S, q)$ is defined to be $\mathsf{Nat}(y^S, q)$ the natural isomorphism $\mathbf{Poly}(y^S, q) \cong q(S)$ follows directly from the Yoneda lemma (Lemma 2.10) with $F = q$.

Solution to Exercise 2.57.

1. Here are the corolla forests associated to $p := y^3 + y + 1$, $q := y^2 + y^2 + 2$, and $r := y^3$ (with each root labeled for convenience).



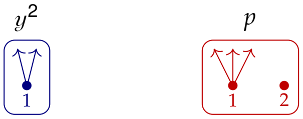
2. Here is one possible morphism $p \to q$ (you may have drawn others).



3. As depicted, our morphism delegates the first decision of p to the second decision of q, whose first and second options are passed back to the third and first options, respectively, of the first decision of p. Then the second decision of p is delegated to the fourth decision of q, which has no options; effectively, the second decision of p has been canceled. Finally, the third decision of p is delegated to the third decision of q, neither of which has any options.
4. There cannot be a morphism $p \to r$ for the following reason: if we delegate the third decision of p, which has no options, to the sole decision of r, which has 3 options, then there is no way to pass a choice of one of the 3 options back to any of the options associated with the third decision of p, as there are no such options.

Solution to Exercise 2.58.

1. We let $p := y^3 + 1$ (you could have selected others) and draw both p and y^2 as corolla forests, labeling each root for convenience.



2. We count all the maps from y^2 to p. The unique position of y^2 can be sent to either p-position. If it is sent to the first p-position, then there are 2 directions of y^2 for each of the 3 $p[1]$-directions to be sent to, for a total of $2^3 = 8$ maps. Otherwise, the unique position of y^2 is sent to the second p-position—at which there are no directions, so there is exactly 1 way to do this. So we have $8 + 1 = 9$ maps from y^2 to p.
3. Yes, the previous answer is equal to $p(2) = 2^3 + 1 = 9$.

Solution to Exercise 2.59.

Our goal is to compute the number of natural transformations $p \to q$ for each of the following polynomials p, q. By (2.54), we always have

$$|\mathbf{Poly}(p,q)| = \prod_{i \in p(1)} |q(p[i])|.$$

1. If $p = y^3$ and $q = y^4$, then
$$|\mathbf{Poly}(p,q)| = \prod_{i \in 1} |p[i]|^4 = 3^4 = 81.$$
2. If $p = y^3 + 1$ and $q = y^4$, then
$$|\mathbf{Poly}(p,q)| = \prod_{i \in 2} |p[i]|^4 = 3^4 \cdot 0^4 = 0.$$

3. If $p = y^3 + 1$ and $q = y^4 + 1$, then
$$|\mathbf{Poly}(p,q)| = \prod_{i \in 2} |p[i]|^4 + 1 = (3^4 + 1)(0^4 + 1) = 82.$$

4. If $p = 4y^3 + 3y^2 + y$ and $q = y$, then
$$|\mathbf{Poly}(p,q)| = \prod_{i \in 8} |p[i]| = 3^4 \cdot 2^3 \cdot 1 = 648.$$

5. If $p = 4y^3$ and $q = 3y$, then
$$|\mathbf{Poly}(p,q)| = \prod_{i \in 4} 3|p[i]| = (3 \cdot 3)^4 = 6561.$$

Solution to Exercise 2.60.

1. We will show that the following isomorphism holds:
$$\mathbf{Poly}(p,q) \cong \prod_{i \in p(1)} \sum_{j \in q(1)} \prod_{d \in q[j]} \sum_{c \in p[i]} 1.$$
By (2.54), it suffices to show that for all $i \in p(1)$ and $j \in q(1)$, we have
$$p[i]^{q[j]} \cong \prod_{d \in q[j]} \sum_{c \in p[i]} 1.$$
Indeed, by (2.26) and Exercise 2.16, we have
$$\begin{aligned}
\prod_{d \in q[j]} \sum_{c \in p[i]} 1 &\cong \sum_{\bar{c} \colon q[j] \to p[i]} \prod_{d \in q[j]} 1 && (2.26)\\
&\cong \sum_{\bar{c} \colon q[j] \to p[i]} 1 && \text{(Exercise 2.16 \#2)}\\
&\cong \mathbf{Set}(q[j], p[i]) && \text{(Exercise 2.16 \#1)}\\
&\cong p[i]^{q[j]}.
\end{aligned}$$

2. We will show that the following isomorphism holds:
$$\mathbf{Poly}(p,q) \cong \sum_{f_1 \colon p(1) \to q(1)} \prod_{j \in q(1)} \mathbf{Set}\left(q[j], \prod_{\substack{i \in p(1),\\ f_1(i) = j}} p[i]\right).$$
By (2.54) and (2.26), we have
$$\begin{aligned}
\mathbf{Poly}(p,q) &\cong \prod_{i \in p(1)} \sum_{j \in q(1)} p[i]^{q[j]} && (2.54)\\
&\cong \sum_{f_1 \colon p(1) \to q(1)} \prod_{i \in p(1)} p[i]^{q[f_1(i)]} && (2.26)\\
&\cong \sum_{f_1 \colon p(1) \to q(1)} \prod_{j \in q(1)} \prod_{\substack{i \in p(1),\\ f_1(i) = j}} p[i]^{q[j]} && (*)\\
&\cong \sum_{f_1 \colon p(1) \to q(1)} \prod_{j \in q(1)} \mathbf{Set}\left(q[j], \prod_{\substack{i \in p(1),\\ f_1(i) = j}} p[i]\right) && \text{(Universal property of products)}
\end{aligned}$$
where $(*)$ follows from the fact that for any function $f_1 \colon p(1) \to q(1)$, the set $p(1)$ can be written as the disjoint union of sets of the form $f_1^{-1}(j) = \{i \in p(1) \mid f_1(i) = j\}$ for each $j \in q(1)$.

3. To explain how the set

$$D_{p,q} \coloneqq \sum_{f_1\colon p(1)\to q(1)} \prod_{j\in q(1)} \mathbf{Set}\left(q[j], \prod_{\substack{i\in p(1),\\ f_1(i)=j}} p[i]\right)$$

specifies a way of delegating decisions from p to q, we first give the instructions for choosing an element of $D_{p,q}$ as a nested list:

```
To choose an element of $D_{p,q}$:
1. choose a function $f_1\colon p(1) \to q(1)$;
2. for each element $j \in q(1)$:
   1. for each element of $q[j]$:
      1. for each element $i \in p(1)$ satisfying $f_1(i) = j$:
         1. choose an element of $p[i]$.
```

So f_1 delegates each of p's decisions to one of q's decisions. Then for every option of every decision j of q, we choose an option of each of p's decisions that has been delegated to j by f_1.

Solution to Exercise 2.63.

Given $q \in \mathbf{Poly}$ and $p_i \in \mathbf{Poly}$ for each $i \in I$ for some set I, we use (2.51) and (2.54) to verify that

$$\begin{aligned}
\mathbf{Poly}\left(\sum_{i\in I} p_i, q\right) &\cong \mathbf{Poly}\left(\sum_{(i,j)\in\sum_{i\in I} p_i(1)} y^{p_i[j]}, q\right) && (2.51)\\
&\cong \prod_{(i,j)\in\sum_{i\in I} p_i(1)} q(p_i[j]) && (2.54)\\
&\cong \prod_{i\in I}\prod_{j\in p_i(1)} q(p_i[j])\\
&\cong \prod_{i\in I} \mathbf{Poly}(p_i, q). && (2.54)
\end{aligned}$$

Solution to Exercise 2.65.

We can evaluate $\dot{p}(1)$ directly from the definition of $\dot{p}$ to obtain

$$\dot{p}(1) = \sum_{i\in p(1)} \sum_{d\in p[i]} 1^{p[i]-\{d\}} \cong \sum_{i\in p(1)} p[i],$$

which is isomorphic to the set of all directions of p. Then $\pi_p\colon \dot{p}(1) \to p(1)$ is the canonical projection, sending each direction d of p to the position i of p for which $d \in p[i]$.

Solution to Exercise 2.66.

Here $p, q \in \mathbf{Poly}$.

1. Our goal is to characterize the canonical map $\dot{p}y \to p$. If we unravel the definitions, this is a map
$$\sum_{i\in p(1)}\sum_{d\in p[i]} y^{p[i]} \to \sum_{i\in p(1)} y^{p[i]}.$$
We observe that there is always such a map sending every position $(i, d) \in \sum_{i\in p(1)} p[i]$ of $\dot{p}y$ to its first projection $i \in p(1)$ and is the identity on directions. This is the canonical map.
2. There cannot always be a canonical map $p \to \dot{p}$, for if $p \coloneqq 1$, then $\dot{p} \coloneqq 0$, and there is no map $1 \to 0$.
3. We show that there cannot always be a canonical map $\dot{p} \to p$. Take $p \coloneqq y$, so $\dot{p} \coloneqq 1$. A map $1 \to y$ must have an on-directions function $1 \to 0$, but such a function does not exist.
4. We show that even when there is a map $p \to q$, there is not necessarily a map $\dot{p} \to \dot{q}$. Take $p \coloneqq y$ and $q \coloneqq 1$. Then there is a map $p \to q$ that sends the unique position of y to the unique position of 1 and is the empty function on directions. But $\dot{p} = 1$ and $\dot{q} = 0$, and there is no map $1 \to 0$.

5. We show that there is a canonical map $g\colon [p, y] \otimes p \to \dot{p}$, where $[p, y] \otimes p$ is given by (2.67). The on-positions function g_1 takes $f \in \prod_{i \in p(1)} p[i]$ and $i \in p(1)$ and sends the pair of them to the $\dot{p}$-position corresponding to $i \in p(1)$ and $f(i) \in p[i]$. We then have $\dot{p}[(i, f(i))] \cong p[i]$ and $([p, y] \otimes p)[(f, i)] \cong p(1) \times p[i]$, so the on-directions function $g^\sharp_{(f,i)}$ can send each $d \in p[i]$ to $(i, d) \in p(1) \times p[i]$.
6. We wish to describe a map $py \to q$ in terms of "unassigned to directions." Observe that as an arena, py has the same positions as p but has one more direction than p does at each position. We denote this extra direction at each position $i \in p(1)$ of py by $*_i$. So a map $f\colon py \to q$ sends each position i of p to a position j of q, but every $q[j]$-direction could either be sent back to a $p[i]$-direction or the extra direction $*_i$. We can say that an arena morphism $f\colon py \to q$ is like an arena morphism $p \to q$, except that any of the directions of q may remain unassigned—i.e. we may have partial on-directions functions.

Solution to Exercise 2.73.

We wish to show that the lens $((\mathrm{id}_p)_1, (\mathrm{id}_p)^\sharp)$ associated to the identity natural transformation id_p of a polynomial p is such that $(\mathrm{id}_p)_1$ and every $(\mathrm{id}_p)^\sharp_i$ are all identity functions. Indeed, by Corollary 2.72, for each $i \in p(1)$, we have

$$((\mathrm{id}_p)_1(i), (\mathrm{id}_p)^\sharp_i) = (\mathrm{id}_p)_{p[i]}(i, \mathrm{id}_{p[i]}) = (i, \mathrm{id}_{p[i]}),$$

as $(\mathrm{id}_p)_{p[i]}$ is the identity function on $p(p[i])$.

Solution to Exercise 2.74.

Given polynomial morphisms $f\colon p \to q, g\colon q \to r$, and their composite $h \coloneqq f \mathbin{⨟} g$, we wish to show that the lens $(h_1, h^\sharp)$ associated to h satisfies $h_1 = f_1 \mathbin{⨟} g_1$ and $h^\sharp_i = g^\sharp_{f_1(i)} \mathbin{⨟} f^\sharp_i$ for all $i \in p(1)$. Indeed, by Corollary 2.72 and Proposition 2.70, for each $i \in p(1)$, we have

$$\begin{aligned}
(h_1(i), h^\sharp_i) &= h_{p[i]}(i, \mathrm{id}_{p[i]}) \\
&= g_{p[i]}(f_{p[i]}(i, \mathrm{id}_{p[i]})) && (h = f \mathbin{⨟} g) \\
&= g_{p[i]}(f_1(i), f^\sharp_i) && (\text{Corollary 2.72}) \\
&= (g_1(f_1(i)), g^\sharp_{f_1(i)} \mathbin{⨟} f^\sharp_i). && (\text{Proposition 2.70})
\end{aligned}$$

Solution to Exercise 2.76.

We provide $\iota\colon p \to p + q$ and $\kappa\colon q \to p + q$ as follows. On positions, they are the canonical inclusions $\iota_1\colon p(1) \to p(1) + q(1)$ and $\kappa_1\colon q(1) \to p(1) + q(1)$; on directions, they are identities. We wish to show that, for $p, q \in$ **Poly**, the polynomial $p + q$ along with ι and κ satisfy the universal property of the coproduct. That is, we must show that for any $r \in$ **Poly** and maps $f\colon r \to p$ and $g\colon r \to q$, there exists a unique map $h\colon r \to p + q$ for which the diagram (2.77) commutes.

We apply Example 2.75. In order for (2.77) to commute, it must commute on positions—that is, the following diagram of sets must commute:

$$\begin{tikzcd}
p(1) \arrow[r, "\iota_1"] \arrow[dr, "f_1"'] & p(1) + q(1) \arrow[d, dashed, "h_1"] & q(1) \arrow[l, "\kappa_1"'] \arrow[dl, "g_1"] \\
& r(1) &
\end{tikzcd} \tag{2.107}$$

But since $p(1) + q(1)$ along with the inclusions ι_1 and κ_1 form the coproduct of $p(1)$ and $q(1)$ in **Set**, there exists a unique h_1 for which (2.107) commutes. Hence h is uniquely characterized on positions. In particular, it must send each $(1, i) \in p(1) + q(1)$ with $i \in p(1)$ to $f_1(i)$ and each $(2, j) \in p(1) + q(1)$ with $j \in q(1)$ to $g_1(j)$.

Moreover, if (2.77) is to commute on directions, then for every $i \in p(1)$ and $j \in q(1)$, the following diagrams of sets must commute:

$$\begin{tikzcd}
p[i] & (p+q)[(1,i)] \arrow[l, "\iota_i^\sharp"'] & & (p+q)[(2,j)] \arrow[r, "\kappa_j^\sharp"] & q[j] \\
& r[f_1(i)] \arrow[ul, "f_i^\sharp"] \arrow[u, dashed, "h^\sharp_{(1,i)}"'] & & r[g_1(j)] \arrow[ur, "g_j^\sharp"'] \arrow[u, dashed, "h^\sharp_{(2,j)}"] &
\end{tikzcd} \tag{2.108}$$

But $(p+q)[(1,i)] \cong p[i]$ and $\iota_i^\sharp$ is the identity, so we must have $h^\sharp_{(1,i)} = f_i^\sharp$. Similarly, $(p+q)[(2,j)] \cong q[j]$ and $\kappa_j^\sharp$ is the identity, so we must have $h^\sharp_{(2,j)} = g_j^\sharp$. Hence h is also uniquely characterized on directions, so it is unique overall. Moreover, we have shown that we can define h on positions so that (2.107) commutes, and that we can define h on directions such that the diagrams in (2.108) commute. As the commutativity of the diagrams in (2.107) and (2.108) together imply the commutativity of (2.77), it follows that there exists h for which (2.77) commutes.

Solution to Exercise 2.78.

1. Given a topological space X, we can define a polynomial p_X whose positions are the points in X and whose directions at $x \in X$ are the open neighborhoods of x. In other words,
$$p_X \coloneqq \sum_{x \in X} y^{\{U \subseteq X \mid x \in U,\, U \text{ open}\}}.$$
2. For every continuous map $f\colon X \to Y$, we give a lens $p_f\colon p_X \to p_Y$. The on-positions function is just f, while for each position $x \in X$ of p_X, the on-directions function $(p_f)_x^\sharp\colon p_Y[f(x)] \to p_X[x]$ sends each open neighborhood U of $f(x)$ to the open neighborhood $f^{-1}(U)$ of x.
3. To show that p_X is functorial in X, it suffices to show that sending continuous maps $f\colon X \to Y$ to their induced lenses $p_f\colon p_X \to p_Y$ preserves identities and composition. First, we show that for any topological space X, the lens p_{id_X}, where id_X is the identity map on X, is an identity morphism. By #2, the on-positions function of p_{id_X} is id_X, and for each $x \in X$ the on-directions function $(p_f)_x^\sharp\colon p_X[x] \to p_X[x]$ sends $U \in p_X[x]$ to $(\text{id}_X)^{-1}(U) = U$. Hence p_{id_X} is the identity on both positions and directions; it follows from Exercise 2.73 that p_{id_X} is an identity morphism.
We now show that for topological spaces X, Y, and Z and continuous maps $f\colon X \to Y$ and $g\colon Y \to Z$, we have $p_f \mathbin{⨾} p_g = p_{f ⨾ g}$. By #2 and Exercise 2.74, the on-positions function of either side is equal to $f ⨾ g$, so it suffices to show that for all $x \in X$,
$$(p_{f⨾g})_x^\sharp = (p_g)_{f(x)}^\sharp \mathbin{⨾} (p_f)_x^\sharp.$$
Again by #2, the left hand side sends each $U \in p_Z[g(f(x))]$ to $(f ⨾ g)^{-1}(U)$, while the right hand side sends U to $f^{-1}(g^{-1}(U))$, but by elementary set theory, these sets are equal.
4. The functor is not full. Consider the spaces $X = 2$ with the indiscrete topology (i.e. the only open sets are the empty set and X) and $Y = 2$ with the discrete topology (i.e. all subsets are open). Then $p_X \cong 2y$ and $p_Y \cong 2y^2$, so our functor induces a function from the set of continuous functions $X \to Y$ to the set of lenses $2y \to 2y^2$. We claim that this function is not surjective: in particular, consider the lens $h\colon 2y \to 2y^2$ that is the identity on positions (and uniquely defined on directions). Then a continuous function $f\colon X \to Y$ that our functor sends to h must also be the identity on the underlying sets of X and Y. But such a function cannot be continuous, as the preimage under f of a singleton set of Y, which is open, would be a singleton set of X, which would not be open. So our functor sends no continuous function $X \to Y$ to h, and therefore is not full.
The functor is, however, faithful: for any spaces X and Y and continuous function $f\colon X \to Y$, we can uniquely recover f from p_f by taking the on-positions function $(p_f)_1$.

Solution to Exercise 2.82.

Given $q \in \mathbf{Poly}$ and $p_i \in \mathbf{Poly}$ for each $i \in I$ for some set I, we use (2.54) to verify that

$$\begin{aligned}
\mathbf{Poly}\left(q, \prod_{i \in I} p_i\right) &\cong \prod_{k \in q(1)} \left(\prod_{i \in I} p_i\right)(q[k]) && (2.54)\\
&\cong \prod_{k \in q(1)} \prod_{i \in I} p_i(q[k])\\
&\cong \prod_{i \in I} \prod_{k \in q(1)} p_i(q[k])\\
&\cong \prod_{i \in I} \mathbf{Poly}(q, p_i). && (2.54)
\end{aligned}$$

Solution to Exercise 2.83.

Given $p_1 \coloneqq y + 1, p_2 \coloneqq y + 2$, and $p_3 \coloneqq y^2$, we compute $\prod_{i \in 3} p_i$ via (2.80) as follows:

$$\begin{aligned}
\prod_{i \in 3} p_i &\cong \sum_{\bar{j} \in \prod_{i \in 3} p_i(1)} y^{\sum_{i \in 3} p_i[\bar{j}(i)]} && (2.80)\\
&\cong \sum_{\bar{j} \colon (i \in 3) \to p_i(1)} y^{p_1[\bar{j}(1)] + p_2[\bar{j}(2)] + p_3[\bar{j}(3)]}\\
&\cong y^{p_1[1]+p_2[1]+p_3[1]} + y^{p_1[1]+p_2[2]+p_3[1]} + y^{p_1[1]+p_2[3]+p_3[1]}\\
&+ y^{p_1[2]+p_2[1]+p_3[1]} + y^{p_1[2]+p_2[2]+p_3[1]} + y^{p_1[2]+p_2[3]+p_3[1]}\\
&\cong y^{1+1+2} + y^{1+0+2} + y^{1+0+2}\\
&+ y^{0+1+2} + y^{0+0+2} + y^{0+0+2}\\
&\cong y^4 + 3y^3 + 2y^2,
\end{aligned}$$

as we might expect from standard polynomial multiplication.

Solution to Exercise 2.86.

1. We compute the product $A_1 y^{B_1} \times A_2 y^{B_2}$ using (2.84):

$$\begin{aligned}
A_1 y^{B_1} \times A_2 y^{B_2} &\cong \left(\sum_{i \in A_1} y^{B_1}\right) \times \left(\sum_{j \in A_2} y^{B_2}\right)\\
&\cong \sum_{i \in A_1} \sum_{j \in A_2} y^{B_1 + B_2}\\
&\cong A_1 A_2 y^{B_1 + B_2}.
\end{aligned}$$

2. We expand the product $\left(\sum_{i \in I} A_i y^{B_i}\right) \times \left(\sum_{j \in J} A_j y^{B_j}\right)$ by applying (2.24), with $I_1 \coloneqq I$ and $I_2 \coloneqq J$:

$$\begin{aligned}
\left(\sum_{i \in I} A_i y^{B_i}\right) \times \left(\sum_{j \in J} A_j y^{B_j}\right) &\cong \prod_{k \in 2} \sum_{i \in I_k} A_i y^{B_i}\\
&\cong \sum_{\bar{i} \in \prod_{k \in 2} I_k} \prod_{k \in 2} A_{\bar{i}(k)} y^{B_{\bar{i}(k)}}\\
&\cong \sum_{i \in I} \sum_{j \in J} A_i y^{B_i} \times A_j y^{B_j}\\
&\cong \sum_{i \in I} \sum_{j \in J} A_i A_j y^{B_i + B_j}
\end{aligned}$$

where the last isomorphism follows from #1.

Solution to Exercise 2.87.

We wish to show that, for $p, q \in \mathbf{Poly}$, the polynomial pq along with the maps $\pi\colon pq \to p$ and $\phi\colon pq \to q$ as described in the text satisfy the universal property of the product. That is, we must show that for any $r \in \mathbf{Poly}$ and maps $f\colon r \to p$ and $g\colon r \to q$, there exists a unique map $h\colon r \to pq$ for which the following diagram commutes:

$$\begin{tikzcd}
r \arrow[r, "g"] \arrow[d, "f"'] \arrow[dr, dashed, "h"] & q \\
p & pq. \arrow[l, "\pi"] \arrow[u, "\phi"']
\end{tikzcd} \tag{2.109}$$

We apply Example 2.75. In order for (2.109) to commute, it must commute on positions—that is, the following diagram of sets must commute:

$$\begin{tikzcd}
r(1) \arrow[r, "g_1"] \arrow[d, "f_1"'] \arrow[dr, dashed, "h_1"] & q(1) \\
p(1) & (pq)(1). \arrow[l, "\pi_1"] \arrow[u, "\phi_1"']
\end{tikzcd} \tag{2.110}$$

But since $(pq)(1) \cong p(1) \times q(1)$ along with the projections π_1 and ϕ_1 form the product of $p(1)$ and $q(1)$ in **Set**, there exists a unique h_1 for which (2.110) commutes. Hence h is uniquely characterized on positions. In particular, it must send each $k \in r(1)$ to the pair $(f_1(k), g_1(k)) \in (pq)(1)$.

Moreover, if (2.77) is to commute on directions, then for every $k \in r(1)$, the following diagram of sets must commute:

$$\begin{tikzcd}
r[k] & & q[g_1(k)] \arrow[ll, "g_k^\sharp"'] \arrow[d, "\phi^\sharp_{(f_1(k),g_1(k))}"] \\
p[f_1(k)] \arrow[u, "f_k^\sharp"] \arrow[rr, "\pi^\sharp_{(f_1(k),g_1(k))}"'] & & (pq)[(f_1(k), g_1(k))]. \arrow[ull, dashed, "h_k^\sharp"']
\end{tikzcd} \tag{2.111}$$

As $(pq)[(f_1(k), g_1(k))] \cong p[f_1(k)] + q[g_1(k)]$ along with the inclusions $\pi^\sharp_{(f_1(k),g_1(k))}$ and $\phi^\sharp_{(f_1(k),g_1(k))}$ form the coproduct of $p[f_1(k)]$ and $q[g_1(k)]$ in **Set**, there exists a unique $h_k^\sharp$ for which (2.111) commutes. Hence h is also uniquely characterized on directions, so it is unique overall. Moreover, we have shown that we can define h on positions so that (2.110) commutes, and that we can define h on directions such that (2.111) commutes. As the commutativity of (2.110) and (2.111) together imply the commutativity of (2.109), it follows that there exists h for which (2.109) commutes.

Solution to Exercise 2.91.

1. We compute the product $A_1 y^{B_1} \otimes A_2 y^{B_2}$ using (2.89):

$$\begin{aligned}
A_1 y^{B_1} \otimes A_2 y^{B_2} &\cong \left(\sum_{i \in A_1} y^{B_1}\right) \otimes \left(\sum_{j \in A_2} y^{B_2}\right) \\
&\cong \sum_{i \in A_1} \sum_{j \in A_2} y^{B_1 \times B_2} \\
&\cong A_1 A_2 y^{B_1 B_2}.
\end{aligned}$$

2. We expand the product $\left(\sum_{i \in I} A_i y^{B_i}\right) \otimes \left(\sum_{j \in J} A_j y^{B_j}\right)$ as follows:

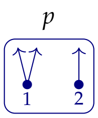
$$\begin{aligned}
\left(\sum_{i \in I} A_i y^{B_i}\right) \otimes \left(\sum_{j \in J} A_j y^{B_j}\right) &\cong \left(\sum_{i \in I} \sum_{i' \in A_i} y^{B_i}\right) \otimes \left(\sum_{j \in J} \sum_{j' \in A_j} y^{B_j}\right) \\
&\cong \sum_{i \in I} \sum_{i' \in A_i} \sum_{j \in J} \sum_{j' \in A_j} y^{B_i \times B_j}
\end{aligned}$$

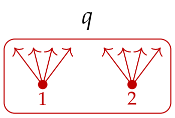
$$
\begin{aligned}
&\cong \sum_{i\in I}\sum_{j\in J}\sum_{i'\in A_i}\sum_{j'\in A_j} y^{B_iB_j}\\
&\cong \sum_{i\in I}\sum_{j\in J} A_iA_j y^{B_iB_j}.
\end{aligned}
$$

Solution to Exercise 2.92.

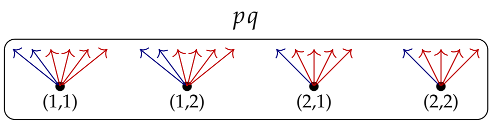
Here $p := y^2 + y$ and $q := 2y^4$.

1. Here are p and q drawn as corolla forests:

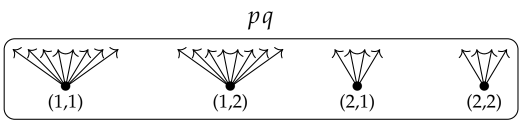




2. Here is pq drawn as a corolla forest:



3. Here is $p \otimes q$ drawn as a corolla forest:



Solution to Exercise 2.93.

Here $p := 2y^2 + 3y$ and $q := y^4 + 3y^3$.

1. We compute $p \times q$ using Exercise 2.86 #2:

$$
\begin{aligned}
p \times q &\cong 2y^{2+4} + (2\times 3)y^{2+3} + 3y^{1+4} + (3\times 3)y^{1+3}\\
&\cong 2y^6 + 6y^5 + 3y^5 + 9y^4\\
&\cong 2y^6 + 9y^5 + 9y^4.
\end{aligned}
$$

2. We compute $p \otimes q$ using Exercise 2.91 #2:

$$
\begin{aligned}
p \otimes q &\cong 2y^{2\times 4} + (2\times 3)y^{2\times 3} + 3y^4 + (3\times 3)y^3\\
&\cong 2y^8 + 6y^6 + 3y^4 + 9y^3.
\end{aligned}
$$

3. We evaluate $(2{\cdot}2^y + 3{\cdot}1^y + 1)\cdot(1{\cdot}4^y + 3{\cdot}3^y + 2)$ using standard high school algebra:

$$
\begin{aligned}
(2{\cdot}2^y + 3{\cdot}1^y)\cdot(1{\cdot}4^y + 3{\cdot}3^y) &= 2{\cdot}1{\cdot}2^y{\cdot}4^y + 2{\cdot}3{\cdot}2^y{\cdot}3^y + 3{\cdot}1{\cdot}1^y{\cdot}4^y + 3{\cdot}3{\cdot}1^y{\cdot}3^y\\
&= 2{\cdot}8^y + 6{\cdot}6^y + 3{\cdot}4^y + 9{\cdot}3^y.
\end{aligned}
$$

4. We describe the connection between the last two parts as follows. Given a polynomial p, we let $d(p)$ denote the Dirichlet series $\sum_{i\in p(1)} |p[i]|^y$. Then by (2.89),

$$
\begin{aligned}
d(p\otimes q) &= \sum_{i\in p(1)}\sum_{j\in q(1)} |p[i]\times q[j]|^y\\
&= \sum_{i\in p(1)} |p[i]|^y \sum_{j\in q(1)} |q[j]|^y\\
&= d(p)\cdot d(q).
\end{aligned}
$$

The last two parts are simply an example of this identity for a specific choice of p and q.

Solution to Exercise 2.94.

We compute $(3y^5 + 6y^2) \otimes 4$ using Exercise 2.91 #2 and the fact that $4 = 4y^0$:

$$\begin{aligned}(3y^5 + 6y^2) \otimes 4y^0 &\cong (3 \times 4)y^{5\times 0} + (6 \times 4)y^{2\times 0}\\ &\cong 12y^0 + 24y^0\\ &\cong 36.\end{aligned}$$

Solution to Exercise 2.95.

1. We show that $p \otimes y \cong p$:

$$\begin{aligned}p \otimes y &\cong \sum_{i\in p(1)} \sum_{j\in 1} y^{p[i]\times 1} && (2.89)\\ &\cong \sum_{i\in p(1)} y^{p[i]} \cong p.\end{aligned}$$

2. We show that $(p \otimes q) \otimes r \cong p \otimes (q \otimes r)$:

$$\begin{aligned}(p \otimes q) \otimes r &\cong \left(\sum_{i\in p(1)} \sum_{j\in q(1)} y^{p[i]\times q[j]}\right) \otimes r && (2.89)\\ &\cong \sum_{i\in p(1)} \sum_{j\in q(1)} \left(\sum_{k\in r(1)} y^{(p[i]\times q[j])\times r[k]}\right) && (2.89)\\ &\cong \sum_{i\in p(1)} \left(\sum_{j\in q(1)} \sum_{k\in r(1)} y^{p[i]\times(q[j]\times r[k])}\right) && (\text{Associativity of } \textstyle\sum \text{ and } \times)\\ &\cong p \otimes \left(\sum_{j\in q(1)} \sum_{k\in r(1)} y^{q[j]\times r[k]}\right) && (2.89)\\ &\cong p \otimes (q \otimes r). && (2.89)\end{aligned}$$

3. We show that $(p \otimes q) \cong (q \otimes p)$:

$$\begin{aligned}p \otimes q &\cong \sum_{i\in p(1)} \sum_{j\in q(1)} y^{p[i]\times q[j]} && (2.89)\\ &\cong \sum_{j\in q(1)} \sum_{i\in p(1)} y^{q[j]\times p[i]} && (\text{Commutativity of } \textstyle\sum \text{ and } \times)\\ &\cong q \otimes p. && (2.89)\end{aligned}$$

Solution to Exercise 2.97.

1. By Exercise 2.91 #1, we have $A \otimes B \cong AB$.
2. We use Exercise 2.91 #2 to compute $A \otimes q$:

$$A \otimes q \cong \sum_{j\in q(1)} Ay^{0\times q[i]} \cong \sum_{j\in q(1)} A \cong A \times q(1).$$

3. By Exercise 2.91 #1, we have $Ay \otimes By \cong ABy$.
4. We show that $(p \otimes q)(1)$ and $p(1) \times q(1)$ are isomorphic. By (2.89),

$$(p \otimes q)(1) \cong \sum_{i\in p(1)} \sum_{j\in q(1)} 1^{p[i]\times q[j]} \cong p(1) \times q(1).$$

Solution to Exercise 2.98.

For each of the following classes of polynomials, we determine whether they are closed under $\otimes$ and whether they contain y.

1. The set $\{Ay^0 \mid A \in \mathbf{Set}\}$ of constant polynomials is closed under $\otimes$ by the solution to Exercise 2.97 #1. But the set does not contain y, as y is not a constant polynomial.
2. The set $\{Ay \mid A \in \mathbf{Set}\}$ of linear polynomials is closed under $\otimes$ by the solution to Exercise 2.97 #3 and does contain y, as $y \cong 1y$.
3. The set $\{Ay + B \mid A, B \in \mathbf{Set}\}$ of affine polynomials is closed under $\otimes$, for Exercise 2.91 #2 yields
$$(Ay + B) \otimes (A'y + B') \cong AA'y + AB' + BA' + BB'.$$
The set contains y, as $y \cong 1y + 0$.
4. The set $\{Ay^2 + By + C \mid A, B, C \in \mathbf{Set}\}$ of quadradic polynomials is not closed under $\otimes$, for even though $y^2 \cong 1y^2 + 0y + 0$ is a quadratic polynomial, Exercise 2.91 #1 implies that
$$y^2 \otimes y^2 \cong y^4,$$
which is not quadratic. The set contains y, as $y \cong 0y^2 + 1y + 0$.
5. The set $\{Ay^B \mid A, B \in \mathbf{Set}\}$ of monomials is closed under $\otimes$ by Exercise 2.91 #1 and does contain y, as $y \cong 1y^1$.
6. The set $\{Sy^S \mid S \in \mathbf{Set}\}$ of systematic polynomials is closed under $\otimes$, for Exercise 2.91 #1 returns
$$Sy^S \otimes Ty^T \cong STy^{ST}.$$
The set contains y, as $y \cong 1y^1$.
7. The set $\{p \in \mathbf{Poly} \mid p(1) \text{ is finite}\}$ is closed under $\otimes$ by the solution to Exercise 2.97 #4. The set contains y, as $y(1) \cong 1$ is finite.

Solution to Exercise 2.99.

The smallest class of polynomials that's closed under $\otimes$ and contains y is just $\{y\}$. This is because by Exercise 2.91 #1, we have $y \otimes y \cong y$.

Solution to Exercise 2.100.

We show that $(p_1 + p_2) \otimes q \cong (p_1 \otimes q) + (p_2 \otimes q)$ using (2.89):
$$\begin{aligned}(p_1 + p_2) \otimes q &\cong \sum_{k \in 2} \sum_{i \in p_k(1)} \sum_{j \in q(1)} y^{p_k[i] \times q[j]} \\ &\cong \sum_{i \in p_1(1)} \sum_{j \in q(1)} y^{p_1[i] \times q[j]} + \sum_{i \in p_2(1)} \sum_{j \in q(1)} y^{p_2[i] \times q[j]} \\ &\cong (p_1 \otimes q) + (p_2 \otimes q).\end{aligned}$$

Solution to Exercise 2.104.

1. To show that the operation $(A, B) \mapsto A + AB + B$ on $\mathbf{Set}$ is associative, we observe that
$$\begin{aligned}(A + AB + B) + (A + AB + B)C + C &\cong A + AB + B + AC + ABC + BC + C \\ &\cong A + AB + ABC + AC + B + BC + C \\ &\cong A + A(B + BC + C) + (B + BC + C).\end{aligned}$$
2. To show that 0 is unital for this operation, we observe that
$$(A, 0) \mapsto A + A0 + 0 \cong A$$
and
$$(0, B) \mapsto 0 + 0B + B \cong B.$$

3. We let $(1, \odot)$ denote the corresponding monoidal product on **Poly** and evaluate $(y^3+y)\odot(2y^2+2)$. By (2.103), with $A \star B \cong A + AB + B$, we have

$$\begin{aligned}
(y^3 + y) \odot (2y^2 + 2) &\cong (y^3 + y^1) \odot (y^2 + y^2 + y^0 + y^0) \\
&\cong y^{3\star 2} + y^{3\star 2} + y^{3\star 0} + y^{3\star 0} + y^{1\star 2} + y^{1\star 2} + y^{1\star 0} + y^{1\star 0} \\
&\cong 2y^{11} + 2y^3 + 2y^5 + 2y^1 .
\end{aligned}$$

Chapter 3

Dynamical systems as polynomial morphisms

Let’s start putting all this **Poly** stuff to use.

3.1 Moore machines

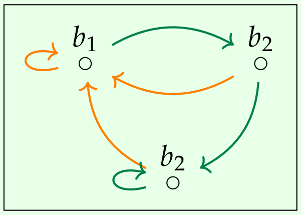
We begin with our simplest example of a dynamical system: a deterministic machine with internal states that can return output and be updated according to input.

Definition 3.1. If A, B, and S are sets, an (A, B)-Moore machine with states S consists of two functions

$$\begin{aligned} \text{return}\colon S &\to B \\ \text{update}\colon S \times A &\to S \end{aligned}$$

We can visualize a Moore machine as follows. It has a set S of possible states. At any point in time, the machine is in one of those states: say $s \in S$. While there, whenever we ask the machine to produce output, it will give us $\text{return}(s) \in B$. But if we feed the machine input $a \in A$, the machine will switch to a new state, $\text{update}(s, a)$. Note that this new state depends not only on the input the machine receives, but also on the state the machine is in when it receives that input.

Example 3.2. Here's a labeled transition diagram of a Moore machine with $S := 3$ states:



(3.3)

Each state is labeled by the output value it returns: an element of $B := \{b_1, b_2\}$. Each state has two outgoing arrows, one orange and one green, representing the two possible inputs, so $A := \{\text{orange}, \text{green}\}$. The targets of the arrows indicate the updated state.

For example, let's say the machine starts at the bottom state. We imagine barking a sequence of inputs—say "orange! orange! green! orange! . . . "—at this machine to make it run through its states and return the output at each state:

1. Starting at the bottom state, the machine returns the output b_2.
2. Receiving the input "orange" at the bottom state, the machine follows the orange arrow from the bottom state to update its state to the left state.
3. At the left state, the machine returns the output b_1.
4. Receiving the input "orange" at the left state, the machine follows the orange arrow from the left state to update its state to—once again—the left state.
5. At the left state, the machine returns the output b_1.
6. Receiving the input "green" at the left state, the machine follows the green arrow from the left state to update its state to the right state.
7. At the right state, the machine returns the output b_2.
8. Receiving the input "orange" at the left state, the machine follows the orange arrow from the right state to update its state to the left state.
9. At the left state, the machine returns the output b_1.

. . .

So given a fixed initial state, namely the bottom state, this Moore machine sends the sequence (orange, orange, green, orange, . . .) of elements in A to the sequence $(b_2, b_1, b_1, b_2, b_1, \ldots)$ of elements in B.

In general, given an initial state $s_0 \in S$, an (A, B)-Moore machine with states S sends every sequence $(a_1, a_2, a_3, \ldots)$ of elements in A to a sequence $(b_0, b_1, b_2, b_3, \ldots)$ of elements in B, defined as follows via an intermediary sequence $(s_0, s_1, s_2, s_3, \ldots)$:

$$b_k := \text{return}(s_k) \qquad \text{and} \qquad s_{k+1} := \text{update}(s_k, a_{k+1})$$

for all $k \in \mathbb{N}$. We'll see that **Poly** gives us a more concise way to express this in **??**.

3.1.1 Moore machines as lenses

Does Definition 3.1 look familiar? It's easy to see that an (A, B)-Moore machine with states S is just a lens between monomials

$$(\text{get}, \text{set})\colon Sy^S \to By^A,$$

with get := return and set := update.[1] Given such a Moore machine, we will call the monomial By^A the *interface*, because it encodes how an outsider can interact with the machine, namely the possible inputs and outputs; and we will call the monomial Sy^S the *state system*.

Exercise 3.4 (Solution here). Write the Moore machine from Example 3.2 as a lens between monomials.

1. What is the state system?
2. What is the interface?

Call the left state 1, the right state 2, and the bottom state 3.

3. What is the on-positions function "get"?
4. What is the on-directions function "set"? ◊

3.1.2 More Moore machines

Example 3.5. There is a dynamical system that takes unchanging input and produces as output the sequence of natural numbers $0, 1, 2, 3, \ldots$. It is a Moore machine with states $\mathbb{N}$ and interface $\mathbb{N}y$. The associated lens $\mathbb{N}y^{\mathbb{N}} \to \mathbb{N}y$ is given by the identity $\mathbb{N} \to \mathbb{N}$ on positions and the function $\mathbb{N} \cong \mathbb{N} \times 1 \to \mathbb{N}$ sending $n \mapsto n + 1$ on directions.

Example 3.6. Here's a(n infinite) Moore machine with states $\mathbb{R}^2$:

$$\mathbb{R}^2 y^{\mathbb{R}^2} \to \mathbb{R}^2 y^{[0,1]\times[0,2\pi)}$$

Its output type is $\mathbb{R}^2$, which we might think of as a location in the 2-dimensional plane, and its input type is $[0, 1] \times [0, 2\pi)$, which we can think of as a command to move a certain distance in a certain direction. The map itself is given by the lens

$$\begin{array}{cc} \mathbb{R}^2 \xrightarrow{\text{return}} \mathbb{R}^2 & \qquad \mathbb{R}^2 \times [0,1] \times [0, 2\pi) \xrightarrow{\text{update}} \mathbb{R}^2 \\ (x, y) \mapsto (x, y) & \qquad (x, y, r, \theta) \mapsto (x + r\cos\theta, y + r\sin\theta) \end{array}$$

[1] Recall from (2.68) that "get" and "set" are our special names for the on-positions and on-directions functions of a lens between monomials.

Exercise 3.7 (Solution here). Explain in words what the Moore machine in Example 3.6 does. ◊

Example 3.8 (From functions to memoryless Moore machines). For any function $f\colon A \to B$, there is a corresponding (A, B)-Moore machine with states B that takes in an element of A and returns the element of B obtained by applying f.

It is given by the map $(\mathrm{id}_B, \pi_A \mathbin{⨾} f)\colon By^B \to By^A$. That is, it is the identity on positions, returning the state directly as output, and on directions it is the function $B \times A \xrightarrow{\pi_A} A \xrightarrow{f} B$, which ignores the current state and applies f to the input to compute the new state.

If the machine begins in state b_0 and is given a sequence $(a_1, a_2, \ldots)$ of elements in A, the machine's outputs will be $(b_0, f(a_1), f(a_2), \ldots)$. We could say this machine is *memoryless*, because at no point does the state of the machine depend on any previous states.

Exercise 3.9 (Solution here). Suppose we have a function $f\colon A \times B \to B$.

1. Find a corresponding (A, B)-Moore machine $By^B \to By^A$.
2. Would you say the machine is memoryless?
3. Now for any function $g\colon A \to B$, give a corresponding (A, B)-Moore machine $By^B \to By^A$ by first turning $g\colon A \to B$ into a function $A \times B \to B$ that discards its second input. ◊

Exercise 3.10 (Solution here). Find $A, B \in \mathbf{Set}$ such that the following can be identified with a morphism $Sy^S \to By^A$, and explain in words what the corresponding Moore machine does (there may be multiple possible solutions):

1. a *discrete dynamical system*, i.e. a set of states S and a transition function $S \to S$ that tells us how to move from state to state.
2. a *magma*, i.e. a set S and a function $S \times S \to S$.
3. a set S and a subset $S' \subseteq S$. ◊

Exercise 3.11 (Solution here). Consider the Moore machine in Example 3.6, and think of it as a robot. Using the terminology from that example, modify the robot as follows.

Add to its state a "health meter," which has a real value between 0 and 10. Make the robot lose half its health whenever it moves to a location whose x-coordinate is negative. Do not output its health; instead, use its health h as a multiplier, allowing it to move a distance of hr given an input of r. ◊

Exercise 3.12 (Tape of a Turing machine; solution here). A Turing machine has a tape. The tape has a cell for each integer, and each cell holds a value $v \in V = \{0, 1, -\}$ of 0,1, or blank. At any given time the tape not only holds this function $f : \mathbb{Z} \to V$ from cell numbers to values, but also a distinguished choice $c \in \mathbb{Z}$ of the "current" cell. Thus the set of states of the tape is $V^{\mathbb{Z}} \times \mathbb{Z}$.

The Turing machine interacts with the tape by asking for the value at the current cell, an element of V, and by telling it to change the value there as well as whether to move left (i.e. decrease the current cell number by 1) or right (i.e. increase by 1). Thus the set of outputs of the tape is V and the set of inputs is $V \times \{L, R\}$.

1. Give the form of the tape as a Moore machine, i.e. lens $t : Sy^S \to By^A$ for appropriate sets S, A, B.
2. Write down the specific t that makes it act like a tape as specified above. ◊

Exercise 3.13 (Solution here). Let's say a file of length n is a function $f : \mathsf{n} \to \mathsf{ascii}$, where $\mathsf{ascii} := \mathsf{256}$. We refer to elements of $\mathsf{n} = \{1, \ldots, n\}$ as entries in the file and, for each entry $i \in \mathsf{n}$, the value $f(i)$ as the character at entry i.

Given a file f, make a file-reading Moore machine whose output type is $\mathsf{ascii} + \{\text{"done"}\}$ and whose input type is

$$\{(s, t) \mid 1 \leq s \leq t \leq n\} + \{\text{"continue"}\}.$$

For any input, if it is of the form (s, t), then the file-reader should go to entry s in the file and read the character at that entry. If the input is "continue," the file-reader should move to the next entry (i.e. from s to $s + 1$) and read that character—unless the new entry would be greater than t, in which case the file-reader should continually output "done" until it receives another (s, t) pair. ◊

While Exercise 3.13 gives us a functioning file-reader, it is a little strange that we are still able to give the input "continue" even when the output is "done," or input a new range of entries before the file-reader has finished reading from the previous range. In Section 3.2, we'll introduce a generalization of Moore machines to handle cases like these, where the array of inputs the machine can receive changes depending on the output that it just returned. In particular, we will be able to let the file-reader "close its port," so that it can't receive signals while it's busy reading, but open its port again once it's "done"; see Example 3.26.

Deterministic state automata

The diagram in Example 3.2 may look familiar to those who have studied automata theory; in fact, a deterministic state automaton can be expressed as a Moore machine (with a preset initial state).

Definition 3.14. A *deterministic state automaton* consists of

1. a set S, elements of which are called *states*;
2. a set A, elements of which are called *input symbols*;
3. a function $u\colon S \times A \to S$, called the *update function*;
4. an element $s_0 \in S$, called the *initial state*; and
5. a subset $F \subseteq S$, called the *accept states*.

Given a sequence (or "word") $(a_1, \ldots, a_n)$ of elements from A (i.e. an element of $\mathsf{List}(A)$, the free monoid on A), we say that the automaton *accepts* this word if starting at the initial state and following the elements in the sequence leads us to an accept state—or, more formally, if the sequence $(s_0, s_1, \ldots, s_n)$ defined

$$s_{k+1} \coloneqq u(s_k, a_{k+1})$$

for all $k \in \mathbb{N}$ is such that s_n is an accept state: $s_n \in F$.

We call any set of words in $\mathsf{List}(A)$ a *language*, and we say that the set of all words that the automaton accepts is the language *recognized* by the automaton.

Remark 3.15. When we study a deterministic state automaton, we are usually interested in which words the automaton accepts and, more generally, what language the automaton recognizes. While intuitive, the condition we provided for when an automaton accepts a word is rather cumbersome to work with. In **??**, we'll see a simpler way of saying whether an automaton accepts a word, as well as specifying what language the automaton recognizes. Better yet, we'll find that this alternative formulation arises naturally from the theory of **Poly**.

Proposition 3.16. A deterministic state automaton with a set of states S and a set of input symbols A can be identified with a pair of maps

$$y \to Sy^S \to \mathsf{2}y^A.$$

Proof. A map $y \to Sy^S$ can be identified with an initial state $s_0 \in S$. A map $Sy^S \to \mathsf{2}y^A$ consists of a function $f\colon S \to \mathsf{2}$, which can be identified with a subset of accept states $F \subseteq S$, together with an update function $u\colon S \times A \to S$. □

But what if we wanted to make a version of this automaton where, whenver the machine hits an accept state, it stops—no longer taking in any inputs? Again, to do this requires a machine whose set of possible inputs is dependent on the output the current state returns. In this case, instead of an update function $u\colon S \times A \to S$, we want an update function that takes in an input $a \in A$ if the state $s \in S$ is *not* an accept state (say, if $f(s) = 1$) but takes in an input in $\mathsf{0}$ (i.e. no input) if the state s *is* an accept state (if $f(s) = 2$). So there is one update function $u_s\colon A \to S$ if $f(s) = 1$, and a different update function $u_s\colon \mathsf{0} \to S$ if $f(s) = 2$. But these are exactly the on-directions functions of a lens $Sy^S \to y^A + 1$. Indeed, replacing our interface monomial with a general polynomial is exactly how we will obtain our generalized dependent Moore machines.

3.2 Dependent dynamical systems

As lenses, each of our Moore machines above has an interface of the form By^A, i.e. a monomial. Every representable summand of the interface has the same exponent A, corresponding to the fact that the set of available inputs into the machine is always A. But by replacing By^A with an arbitrary polynomial p—a sum of monomials, in which different representable summands may have different exponents—we will allow our input set to change.

Definition 3.17 (Dependent dynamical systems)**.** A *dependent dynamical system* (or a *dependent Moore machine*, or simply a *dynamical system*) is a lens

$$\phi\colon Sy^S \to p$$

for some $S \in \mathbf{Set}$ and $p \in \mathbf{Poly}$. The set S is called the set of *states*—with Sy^S called the *state system*—and the polynomial p is called the *interface*. Positions of the interface are called *outputs*, and directions of the interface are called *inputs*.

The lens's on-positions function $\phi_1\colon S \to p(1)$ is called the *return function*, and for each $s \in S$, the lens's on-directions function $\phi^\sharp_s\colon p[\phi_1(s)] \to S$ is called the *update function* at s.

3.2.1 Thinking about dependency

The current output i of a dependent dynamical system with interface p is a p-position: an element of the set $p(1)$. Then any input provided to the update function at that time must come from the direction-set $p[i]$. So the set of available inputs varies depending on the current output. What kind of system has that kind of a relationship between its inputs and outputs?

Think back to our promises for more generalized dynamical systems from Section 1.2. We can begin to think of outputs not only as outward expressions, but as positions that one takes within one's arena—terminology we've been using all along. If the arena is your body, then your outputs would be the positions that your body could take. This includes where you go, as well as the direction you're looking with your head and eyes, whether your lips are pursed or not, etc. In fact, every form of output you provide, from talking to gesturing to moving another object, is performed by changing your position. And your position determines what inputs you'll notice: if your eyes are closed, your input type is different than if your eyes are open.

The position you're in is itself a sensory organ: a hand outstretched, an eye open or closed.

If we squint, we could even see an output more as a sensing apparatus than anything else. This is pretty philosophical, but imagine your outputs—what you say and do—are more there as a question to the world, a way of sensing what the world is like.

Remark 3.18. It may seem limiting that the set of possible inputs of a dependent dynamical system should depend on the current *output*, rather than the current *state*. Isn't it possible to have a system with different states that give the same output but take in inputs from different sets?

Philosophically, the answer to this question is "no" as long as we think about the interface of a dynamical system as capturing everything about how the system interacts with the outside world. In particular, the output of a system should capture everything an external observer could possibly perceive about the system, while the input set should capture every way an external force can independently decide to interact with the system.

But if the range of ways in which an external agent can choose to interact with the system *changes*, the external agent should be able to detect this fact! If the internal state changes, but the external output remains the same, the agent wouldn't see any difference—they wouldn't know to interact with the system any differently, so their range of possible inputs would have to stay the same, too. That's why it makes sense for the set of possible inputs to depend not on the hidden internal state, but on the output currently exposed.

But however you think of dependent dynamical systems, we need to get a feel for how they work mathematically. We'll do this by going through several examples.

3.2.2 Examples of dependent dynamical systems

To start, let's finish up the example from the end of the last section.

Example 3.19. Recall deterministic state automata from Definition 3.14. Say we want such an automaton to halt after reaching an accept state and no longer take input. For that, rather than use a map $Sy^S \to 2y^A$, we could use a map

$$\phi\colon Sy^S \to y^A + 1.$$

To give such a map, we provide a return function $\phi_1\colon S \to 2$ (here we are thinking of 2 as the position-set of $y^A + 1$). A function ϕ_1 sends some elements of S to 1 and others to 2; those that are sent to 2 are said to be accept states.

If we reach an accept state, we want the machine to halt. So at the position 2, corresponding to the summand 1, there are no directions—and thus no inputs available. In other words, the update function $\phi_s^\sharp$ is trivial when $\phi_1(s) = 2$.

On the other hand, $\phi_s^\sharp$ is not trivial on those elements $s \in S$ for which $\phi_1(s) = 1$. Instead, these update functions $\phi_s^\sharp\colon A \to S$ specify how the machine updates its state on each input from A, if the current state is s. This is in line with the way the automaton's update function behaves.

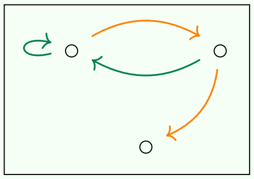
When equipped with an initial state $s_0 \in S$ specified by a map $y \to Sy^S$, we call these dependent dynamical systems *halting deterministic state automata*. Given a sequence (or "word") $(a_1, \ldots, a_n)$ in $\mathsf{List}(A)$, we say that the automaton *accepts* this word if starting at the initial state and following the elements in the sequence leads us to an accept state, *without hitting an accept state and stopping too early*—or, more formally, if there exists a sequence $(s_0, s_1, \ldots, s_n)$ such that $\phi_1(s_n) = 2$ and, for all $k \in [0, n-1]$, we have $\phi_1(s_k) = 1$ and

$$s_{k+1} = \phi^\sharp_{s_k}(a_{k+1}).$$

We call the set of all words accepted by the automaton the language *recognized* by the automaton.

Remark 3.20. Again, the conditions for when one of these automata accepts a word are rather awkward to formally state. We will see in **??** a nicer way of saying whether a word is accepted or a language is recognized by a halting deterministic state automaton.

Exercise 3.21 (Solution here). Consider the halting deterministic state automaton shown below:



(3.22)

Call the left state 1, the right state 2, and the bottom state 3. We designate state 1 as the initial state. We can also call the orange arrows "orange" and the green arrows "green." Answer the following questions, in keeping with the notation from Example 3.19.

1. What is S?
2. What is A?
3. Based on the labeled transition diagram, which states are accept states, and which are not?
4. Specify the corresponding lens $Sy^S \to y^A + 1$.
5. Name a word that is accepted by this automaton.
6. Name a word that is not accepted by this automaton. Why not? Can you find another word that is not accepted by this automaton for a different reason? ◊

Every graph gives rise to a dynamical system—but to ensure that we are talking about the same thing, let us fix the definition of a graph.

Definition 3.23. A *graph* $G = (E \rightrightarrows V)$ consists of an edge set E, a vertex set V, a source function $s\colon E \to V$, and a target function $t\colon E \to V$.

So when we say "graph," we mean a *directed* graph, and we allow multiple edges between the same pair of vertices as well as self-loops.

Example 3.24 (Graphs as dynamical systems). Given a graph $G = (E \rightrightarrows V)$ with source map $s\colon E \to V$ and target map $t\colon E \to V$, there is an associated polynomial

$$g := \sum_{v \in V} y^{s^{-1}(v)}.$$

Its positions are the vertices of the graph, and at each position $v \in V$, its directions are the edges coming out of v. We call this the *emanation polynomial* of G.

The graph itself can be seen as a dynamical system $\phi\colon Vy^V \to g$, where $\phi_1 = \mathrm{id}_V$ and $\phi^\sharp_v(e) = t(e)$. So its states are the vertices of the graph, each vertex returns itself as output, and an input at a vertex $v \in V$ is an edge $e \in E$ coming out of that vertex that takes us from the vertex $v = s(e)$ along the edge e to its target vertex $\phi^\sharp_v(e) = t(e)$.

Exercise 3.25 (Solution here). Pick your favorite graph G, and consider the associated dynamical system as in Example 3.24. Draw its labeled transition diagram as in (3.3) or (3.22). ◊

Example 3.26. In Exercise 3.13 one is tasked with building a file-reader as a Moore machine, where a file is a function $f\colon \mathsf{n} \to \mathsf{ascii}$. Now we turn that file-reader into a dependent dynamical system $\phi\colon Sy^S \to p$ that cannot take in input while it is reading.

We let $S := \{(s,t) \mid 1 \leq s \leq t \leq n\}$, so that each state consists of a current entry s and a terminal entry t. Meanwhile, our interface p will have two labeled copies of ascii as positions:

$$p(1) := \{\text{'ready'}, \text{'busy'}\} \times \mathsf{ascii}.$$

So each p-position is a pair (m, c), where $c \in \mathsf{ascii}$ and m is one of two modes: 'ready' or 'busy.' These form the possible outputs of the file-reader. As for the inputs, we define the direction-sets of p as follows, for all $c \in \mathsf{ascii}$:

$$p[(\text{'ready'}, c)] := S \qquad \text{and} \qquad p[(\text{'busy'}, c)] := 1.$$

That way, our file-reader can receive any pair of entries in S as input when it is 'ready,' but can only be told to advance when it is 'busy.'

We want our file-reader to be 'ready' if its current entry is the terminal entry; otherwise, it will be 'busy.' In either case, it will return the ascii character at the current position. So we define the return function ϕ_1 such that, for all $(s,t) \in S$,

$$\phi_1(s,t) = \begin{cases} (\text{'ready'}, f(s)) & \text{if } s = t \\ (\text{'busy'}, f(s)) & \text{otherwise} \end{cases}$$

While the file-reader is 'ready,' we want it to set its new current and terminal entries to be the input. So for each $(s,t) \in S$ for which $s = t$, we define the update function $\phi^\sharp_{(s,t)} \colon S \to S$ to be the identity on S.

On the other hand, while the file-reader is 'busy,' we want it to step forward through the file each time it receives an input. So for each $(s,t) \in S$ for which $s \neq t$, we let the update function $\phi^\sharp_{(s,t)} \colon 1 \to S$ specify the element $(s+1,t) \in S$, thus shifting its current entry up by 1.

Exercise 3.27 (Solution here). Say instead of a file-reader, we wanted a file-searcher, which acts just like the file-reader from Example 3.26 except that it only emits output $c \in \mathsf{ascii}$ when c is a specific character—say, $c = 100$. Give the lens for this file-searcher by explicitly defining its return (on-positions) and update (on-directions) functions. Hint: You should be able to use the same state system. ◊

In the previous exercise, we manually constructed a file-searcher that acted very much like a file-reader. In Exercise 3.38, we'll see a simpler way to construct a file-searcher by leveraging the file-reader we have already defined. Moreover, this construction highlights precisely how our file-searcher is related to our file-reader. This will be possible using *wrapper interfaces*, which we'll introduce in Section 3.3.3.

Example 3.28. Choose $n \in \mathbb{N}$, a *grid size*, and for each $i \in \mathsf{n}$, let D_i be the set

$$D_i \coloneqq \begin{cases} \{0, +1\} & \text{if } i = 1 \\ \{-1, 0, +1\} & \text{if } 1 < i < n \\ \{-1, 0\} & \text{if } i = n \end{cases}$$

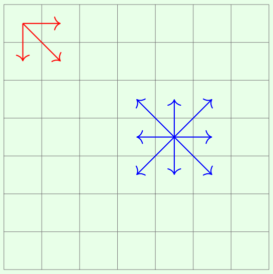
We can think of D_i as the set of all directions a robot could move if at position i. In particular, a robot already at $i = 1$ cannot move in the -1 direction; likewise, a robot already at $i = n$ cannot move in the $+1$ direction.

Then we can model a robot that can be instructed to move within an $\mathsf{n} \times \mathsf{n}$ grid as a dependent dynamical system $\phi \colon Sy^S \to p$, with $S \coloneqq \mathsf{n} \times \mathsf{n}$ and

$$p \coloneqq \sum_{(i,j) \in \mathsf{n} \times \mathsf{n}} y^{D_i \times D_j}.$$

The robot's state is a position in the grid. We let $\phi_1 \coloneqq \mathrm{id}_{\mathsf{n} \times \mathsf{n}}$ so that each state returns itself as output—in particular, the robot returns a position as output by moving to that position in the grid.

Then for each $(i,j) \in \mathsf{n} \times \mathsf{n}$, we let $\phi^\sharp_{(i,j)}$ send each pair of directions $(d,e) \in D_i \times D_j$ to the grid position given by $(i+d, j+e)$. Concretely, this says that if a robot at position (i,j) in the grid receives the pair of directions (d,e) as its input, its new position in the grid will be $(i+d, j+e)$. Our definition of D_i for each $i \in \mathsf{n}$ guarantees that this position is still in the grid. With this setup, the robot has more movement options when it is in the center of the grid than when it is on the sides or corners:



Note that, in this example, the positions of p are literally the positions in the grid where the robot could be, and the directions of p at each position are literally the directions in which the robot can move!

Exercise 3.29 (Solution here). Modify the dynamical system from Example 3.28 as follows.

1. Replace p with another interface p' so that at each grid value, the robot can receive not only the direction it should move in but also a "reward value" $r \in \mathbb{R}$.
2. Replace S with another set of states S' so that an element $s \in S'$ may include both the robot's position and a list of all reward values so far.
3. With your new p' and S', define a new lens $\phi' \colon S'y^{S'} \to p'$ that preserves the behavior of $\phi \colon Sy^S \to p$ from Example 3.28, but also properly updates the robots list of rewards. ◊

In the previous exercise, we added a reward system to the robot on the grid by manually redefining the associated lens. But there is a much simpler way to think about the new system as the juxtaposition of two systems, a robot system and a reward system, in parallel. We'll see how to express this in terms of lenses in Exercise 3.35, once we explain how to juxtapose systems like this in general in Section 3.3.2. In fact, we'll see in Exercise 3.36 that the robot-on-a-grid system itself can be viewed as the juxtaposition of two systems, and this perspective will provide a structured way to generalize Example 3.28 to more than two dimensions.

3.3 Constructing new dynamical systems from old

We have now seen how dependent dynamical systems can be modeled as lenses in **Poly** of the form $Sy^S \to p$. But we have yet to take full advantage of the categorical structure that **Poly** provides. In particular, purely based on what we know of **Poly** so far from Chapter 2, we have three rather different ways of obtaining new dynamical systems from old ones:

1. Given dynamical systems $Sy^S \to p$ and $Sy^S \to q$, we can use the universal property of the *categorical product* to obtain a dynamical system $Sy^S \to p \times q$; see Section 3.3.1.
2. Given dynamical systems $\phi\colon Sy^S \to p$ and $\psi\colon Ty^T \to q$, we can take their parallel product to obtain a dynamical system $\phi \otimes \psi\colon STy^{ST} \to p \otimes q$; see Section 3.3.2.
3. Given a dynamical system $\phi\colon Sy^S \to p$ and a lens $f\colon p \to q$, we can compose them to obtain a dynamical system $\phi \mathbin{⨾} f\colon Sy^S \to q$; see Section 3.3.3.

Each of these operations has a concrete interpretation in terms of dynamical systems. In this section, we'll review each of them in turn.

3.3.1 Categorical products: multiple interfaces operating on the same states

For $n \in \mathbb{N}$, say that we have n dynamical systems, $\phi_i\colon Sy^S \to p_i$ for each $i \in \mathsf{n}$, that all share the same state system. Then by the universal property of products in **Poly**, there is an induced lens

$$\phi\colon Sy^S \to \prod_{i \in \mathsf{n}} p_i,$$

which is itself a dynamical system with state system Sy^S.

We can characterize the dynamics of ϕ in terms of each original dynamical system ϕ_i as follows. By Exercise 2.87, the return function

$$\phi_1\colon S \to \prod_{i \in \mathsf{n}} p_i(1)$$

sends each state $s \in S$ to the corresponding n-tuple of outputs $((\phi_i)_1(s))_{i \in \mathsf{n}}$ returned by each of the original dynamical systems at that state. Then at each state $s \in S$, the update function

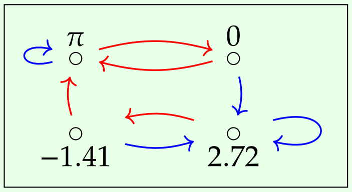
$$\phi^\sharp_s\colon \sum_{i \in \mathsf{n}} p_i[(\phi_i)_1(s)] \to S$$

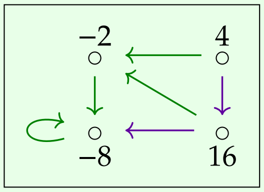
sends each pair (i, d) in its domain, with $i \in \mathsf{n}$ and $d \in p_i[(\phi_i)_1(s)]$, to where the update function of ϕ_i at s sends d: namely $(\phi_i)^\sharp_s(d)$.

In other words, if there are multiple interfaces that can drive the same set of states, we may view them as a single product interface that can drives those states. This single dynamical system returns output in all of the original systems at once; then it can receive input from any one of the original systems' input sets and update its state accordingly. It's as though each of the dynamical systems can see where the combined system is at any time, but only one of them can actually operate it at any given time. So products give us a universal way to combine multiple polynomial interfaces into one.

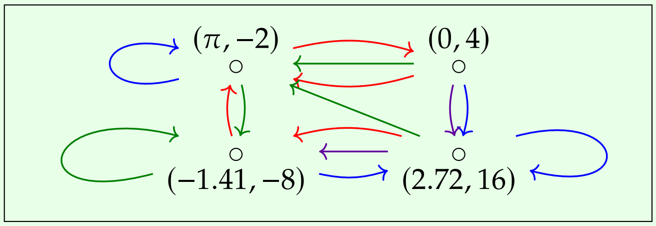
Exercise 3.30 (Solution here). Given $n \in \mathbb{N}$, suppose we have an (A_i, B_i)-Moore machine with state set S for each $i \in \mathsf{n}$. Show that there is an induced $(\sum_{i \in \mathsf{n}} A_i, \prod_{i \in \mathsf{n}} B_i)$-Moore machine, again with state set S. ◊

Example 3.31. Consider two four-state dependent dynamical systems $\phi\colon 4y^4 \to \mathbb{R}y^{\{r,b\}}$ and $\psi\colon 4y^4 \to \mathbb{Z}_{\geq 0}y^{\{g,p\}} + \mathbb{Z}_{<0}y^{\{g\}}$, drawn below as labeled transition diagrams (we think of r, b, g, and p as red, blue, green, and purple, respectively):





The universal property of products provides a unique way to put these systems together to obtain a dynamical system $4y^4 \to \mathbb{R}\mathbb{Z}_{\geq 0}y^{\{r,b,g,p\}} + \mathbb{R}\mathbb{Z}_{<0}y^{\{r,b,g\}}$ that looks like this:



Each state now returns two outputs: one according to the return function of ϕ, and another according to the return function of ψ. As for the possible inputs, we now have the option of giving either an input from a direction-set of ϕ (either r or b), in which case the dynamical system will update its state according to the corresponding update function of ϕ, or an input from a direction-set of ψ (either g or sometimes p), in which case the dynamical system will update its state according to the corresponding update function of ψ.

Exercise 3.32 (Toward event-based systems; solution here). Let $\phi\colon Sy^S \to p$ be a dynamical system. It is constantly needing input at each time step. An event-based system is one that doesn't always get input, and only reacts when it does.

So suppose we want to allow our dynamical system not to do anything. That is, rather than needing to press a button corresponding to a direction of p at each time step, we want to be able to *not* press any button, in which case the system just stays where it is. We want a new system $\phi'\colon Sy^S \to p'$ that has this behavior; what are p' and ϕ'? ◊

3.3.2 Parallel products: juxtaposing dynamical systems

Another way to combine two polynomials—and indeed two lenses—is by taking their parallel product, as in Section 2.4.2. In particular, the parallel product of two state systems is still a state system. So parallel products give us another way to create new dynamical systems from old ones. In fact, it's about as easy as you could hope: you just multiply the set of states, multiply the set of outputs, and multiply the set of inputs at each of those outputs.

For $n \in \mathbb{N}$, say that we have n dynamical systems, $\phi_i \colon S_i y^{S_i} \to p_i$ for every $i \in \mathsf{n}$. Then we can take the parallel product of all of them to get a lens

$$\phi \colon \left(\prod_{i \in \mathsf{n}} S_i\right) y^{\prod_{i \in \mathsf{n}} S_i} \cong \bigotimes_{i \in \mathsf{n}} S_i y^{S_i} \to \bigotimes_{i \in \mathsf{n}} p_i,$$

which is itself a dynamical system.

We can characterize the dynamics of ϕ in terms of each constituent dynamical system ϕ_i as follows. By the proof of Proposition 2.96, the return function

$$\phi_1 \colon \prod_{i \in \mathsf{n}} S_i \to \prod_{i \in \mathsf{n}} p_i(1)$$

sends each n-tuple of states $(s_i)_{i \in \mathsf{n}}$ in its domain, with each $s_i \in S_i$, to the n-tuple of outputs $((\phi_i)_1(s_i))_{i \in \mathsf{n}}$ returned by each of the constituent dynamical systems at each state. Then at the n-tuple of states $(s_i)_{i \in \mathsf{n}} \in \prod_{i \in \mathsf{n}} S_i$, the update function

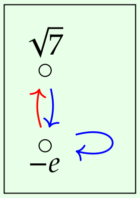
$$\phi^\sharp_{(s_i)_{i \in \mathsf{n}}} \colon \prod_{i \in \mathsf{n}} p_i[(\phi_i)_1(s_i)] \to \prod_{i \in \mathsf{n}} S_i$$

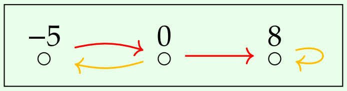
sends each n-tuple of directions $(d_i)_{i \in \mathsf{n}}$ in its domain, with each $d_i \in p_i[(\phi_i)_1(s_i)]$, to the n-tuple $((\phi_i)^\sharp_{s_i}(d_i))_{i \in \mathsf{n}}$ consisting of states where the update function of each ϕ_i at s_i sends d_i.

In other words, multiple dynamical systems running in parallel can be thought of as a single dynamical system. This system stores the states of all the constituent systems at once and returns output from all of them together; then it can receive input from all of the constituent systems' input sets at once and update each constituent state accordingly. So parallel products give us a way to juxtapose multiple dynamical systems in parallel to form a single system.

Exercise 3.33 (Solution here). Given $n \in \mathbb{N}$, suppose we have an (A_i, B_i)-Moore machine with state set S_i for every $i \in \mathsf{n}$. Show that there is an induced $(\prod_{i \in \mathsf{n}} A_i, \prod_{i \in \mathsf{n}} B_i)$-Moore machine with state set $\prod_{i \in \mathsf{n}} S_i$. ◊

Example 3.34. Consider two dependent dynamical systems $\phi \colon 2y^2 \to \mathbb{R}_{<0} y^{\{b,r\}} + \mathbb{R}_{\geq 0} y^{\{b\}}$ and $\psi \colon 3y^3 \to \mathbb{Z}_{<0} y^{\{r\}} + \{0\} y^{\{r,y\}} + \mathbb{Z}_{>0} y^{\{y\}}$, drawn below as labeled transition diagrams (we think of b, r, and y as blue, red, and yellow, respectively):

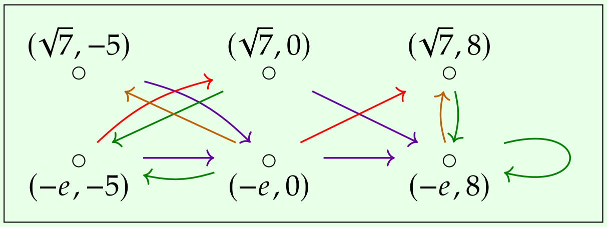




Taking their parallel product, we obtain a dynamical system with state system $\mathsf{6}y^{\mathsf{6}}$ and interface

$$\mathbb{R}_{<0}\mathbb{Z}_{<0}y^{\{(b,r),(r,r)\}} + \mathbb{R}_{<0}\{0\}y^{\{(b,r),(b,y),(r,r),(r,y)\}} + \mathbb{R}_{<0}\mathbb{Z}_{>0}y^{\{(b,y),(r,y)\}}$$
$$+ \mathbb{R}_{\geq 0}\mathbb{Z}_{<0}y^{\{(b,r)\}} + \mathbb{R}_{\geq 0}\{0\}y^{\{(b,r),(b,y)\}} + \mathbb{R}_{\geq 0}\mathbb{Z}_{>0}y^{\{(b,y)\}}$$

that looks like this (we use purple to indicate (b, r), red to indicate (r, r), green to indicate (b, y), and orange to indicate (r, y)):



Each state—really a pair of states from the constituent state sets—returns two outputs, one according to the return function of ϕ and another according to the return function of ψ. Then every input must be a pair of inputs from the constituent state sets, with the update function updating each state in the pair given each input in the pair according to the constituent update functions ϕ and ψ.

Exercise 3.35 (Solution here). Explain how the dynamical system $\phi' \colon S'y^{S'} \to p'$ you built in Exercise 3.29 can be expressed as the parallel product of the robot-on-a-grid dynamical system $\phi \colon Sy^S \to p$ from Example 3.28 with another dynamical system, $\psi \colon Ty^T \to q$. Be sure to specify T, q, and ψ. ◊

Exercise 3.36 (Solution here).

1. Explain how the robot-on-a-grid dynamical system $\phi \colon Sy^S \to p$ from Example 3.28 can be written as the parallel product of some dynamical system with itself.
2. Use k-fold parallel products to generalize Example 3.28 to robots on k-dimensional grids. ◊

The parallel product takes two dynamical systems and essentially puts them in the same room together so that they can be run at the same time. But it doesn't allow for any interaction *between* the two systems. For that, we will need to use what we call a wrapper interface. We'll introduce wrapper interfaces in the next section, before describing how they can be used in conjunction with parallel products to model general interaction in Section 3.4.

3.3.3 Composing lenses: wrapper interfaces

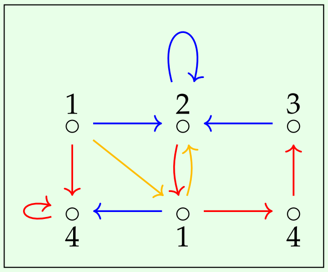
The idea behind a wrapper interface is simple. Given a dynamical system $\phi\colon Sy^S \to p$, say that we wanted to interact with it using a new interface q rather than p. We can do this as long as we have a lens $f\colon p \to q$, which we can then compose with our original dynamical system to obtain a new system $Sy^S \xrightarrow{\phi} p \xrightarrow{f} q$. We call the lens f the *wrapper* and its codomain q the *wrapper interface*, which we *wrap* around ϕ (or sometimes just p, if a dynamical system ϕ has yet to be specified) using f.

To see how this new composite system $\phi \mathbin{⨾} f$ relates to the original dynamical system ϕ, it is best to view f as a delegation of decisions like we did in Section 2.3.2. The lens f converts an output i from p to an output $f_1(i)$ from q on positions, but at the same time is allowing the decision of which input in $p[i]$ to choose next to depend on a choice of input from $q[f_1(i)]$ instead, via its on-directions function at i. So the wrapper f converts output from the original interface p to output from the wrapper interface q, and it converts input into the wrapper interface q to input into the original interface p. Precomposed with a dynamical system, we obtain a new dynamical system that allows you to interact with the original system using only this new interface wrapped around it.

Example 3.37. Consider a dependent dynamical system $\phi\colon \mathsf{6}y^{\mathsf{6}} \to p$ with

$$p \coloneqq \{1\}y^{\{b,y,r\}} + \{2\}y^{\{b,r\}} + \{3\}y^{\{b\}} + \{4\}y^{\{r\}},$$

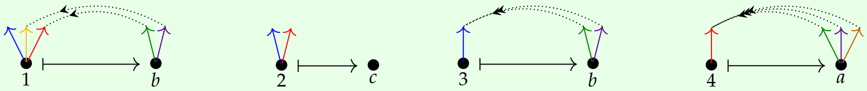
drawn below as a labeled transition diagram (we think of b, y, and r as blue, yellow, and red, respectively):



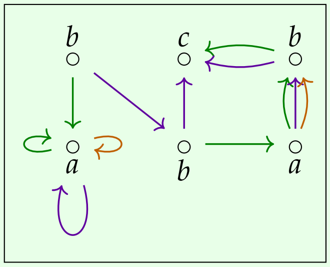
We will wrap the interface

$$q \coloneqq \{a\}y^{\{g,p,o\}} + \{b\}y^{\{g,p\}} + \{c\}$$

around ϕ using the following lens $f\colon p \to q$ (we think of g, p, and o as green, purple, and orange, respectively):



Composing ϕ with f, we obtain a dynamical system $6y^6 \xrightarrow{\phi} p \xrightarrow{f} q$ that looks like this:



Each state returns an output in q according to where the on-positions function of f sends the output the state returns in p. Then each input in q is passed to an input in p via the corresponding on-directions function of f, whereupon the update function of ϕ computes the new state. So f allows us to operate ϕ with the wrapper interface q instead of the original interface p.

Exercise 3.38 (Solution here). In Exercise 3.27, you constructed a file-searcher $\psi\colon Sy^S \to q$ by taking the file-reader $\phi\colon Sy^S \to p$ from Example 3.26 and replacing its interface p with a new interface q while keeping its state system Sy^S the same. Express this construction as wrapping q around ϕ by giving a lens $f\colon p \to q$ for which composing ϕ with f yields ψ. ◊

In the next section, we describe a special kind of wrapper.

3.3.4 Situations as enclosures

Say we wanted to model a dynamical system $\phi\colon Sy^S \to p$ within a closed system, for which an external agent can perceive no change in output and effect no change in input. We can think of this as wrapping y, the interface with one output and one input, around ϕ. To do so, we must specify a wrapper $\gamma\colon p \to y$. We call such a wrapper an *enclosure* for an interface p, since it is a way of closing off p to the outside world.

Let us zoom out from the dynamical systems interpretation of polynomials for the time being to examine the categorical properties of enclosures. Given a polynomial p, we denote the set of lenses $p \to y$ by

$$\Gamma(p) := \mathbf{Poly}(p, y) \tag{3.39}$$

as we did in Proposition 1.22. By (2.54), we have that

$$\Gamma(p) \cong \prod_{i \in p(1)} p[i], \tag{3.40}$$

so in terms of arenas, a map $\gamma \in \Gamma(p)$ can be thought of as a dependent function that assigns each p-position i to a direction $\gamma(i)$ of p at i. So we call an element of $\Gamma(p)$ a *situation* for p. The idea is that the situation you're in gives you a direction to go along at any position you may take.

Returning to the language of dynamical systems, a situation for p corresponds to an enclosure for the interface p, in that it chooses a fixed input at every output of p. An enclosure for your interface dictates what you'll see (the input you receive) given anything you might do (the output you provide); there is no need for any further outside interference.

Remark 3.41. Although they both refer to lenses into y, we will use the term *enclosure* in the context of dynamical systems and wrappers and *situation* more generally.

Exercise 3.42 (Solution here). The notation $\Gamma(p)$ for the set of situations for a polynomial p comes from the common mathematical concept of a "global section." Show that situations $\gamma\colon p \to y$ are precisely *sections* (as defined in Exercise 2.18) of the canonical function $\pi_p\colon \dot{p}(1) \to p(1)$ from Exercise 2.65. ◊

Exercise 3.43 (Solution here). Let $\phi \to Sy^S \to By^A$ be an (A, B)-Moore machine.

1. Is it true that an enclosure $\gamma\colon By^A \to y$ can be identified with a function $A \to B$?
2. Describe how to interpret an enclosure $\gamma\colon By^A \to y$ as a wrapper around an interface By^A.
3. Given an enclosure γ, describe the dynamics of the composite Moore machine $Sy^S \xrightarrow{\phi} By^A \xrightarrow{\gamma} y$ obtained by wrapping y around ϕ using γ. ◊

Proposition 3.44. The situations functor $\Gamma\colon \textbf{Poly} \to \textbf{Set}^{\text{op}}$ sends $(0, +)$ to $(1, \times)$:

$$\Gamma(0) \cong 1 \qquad \text{and} \qquad \Gamma(p + q) \cong \Gamma(p) \times \Gamma(q).$$

Technically, one could say that Γ preserves coproducts, since coproducts in $\textbf{Set}^{\text{op}}$ are products in $\textbf{Set}$.

Exercise 3.45 (Solution here). Prove Proposition 3.44. ◊

Remark 3.46. The situations functor $\Gamma\colon \textbf{Poly} \to \textbf{Set}^{\text{op}}$ is also normal lax monoidal in the sense that there are canonical functions

$$1 \cong \Gamma(y) \qquad \text{and} \qquad \Gamma(p) \times \Gamma(q) \to \Gamma(p \otimes q)$$

satisfying certain well-known laws. But we won't need this, so we omit its proof.

3.4 General interaction

We now have all the pieces we need to fulfill the promises of Section 1.2 by modeling interactions between dependent dynamical systems, which can change their interfaces and interaction patterns, using **Poly**.

3.4.1 Wrapping juxtaposed dynamical systems together

When wrapper interaces are used in conjunction with parallel products, they may encode multiple interacting dynamical systems as a single system. Explicitly, given $n \in \mathbb{N}$ and n dynamical systems, $\phi_i \colon S_i y^{S_i} \to p_i$ for every $i \in \mathsf{n}$, we can first juxtapose them into a single dynamical system

$$\phi\colon \left(\prod_{i \in \mathsf{n}} S_i\right) y^{\prod_{i \in \mathsf{n}} S_i} \to \bigotimes_{i \in \mathsf{n}} p_i$$

by taking their parallel product. Then we can wrap an interface q around ϕ using a wrapper f, yielding a new dynamical system

$$\left(\prod_{i \in \mathsf{n}} S_i\right) y^{\prod_{i \in \mathsf{n}} S_i} \xrightarrow{\phi} \bigotimes_{i \in \mathsf{n}} p_i \xrightarrow{f} q.$$

On positions, f gives a way of combining all the outputs of the constituent interfaces into a single output of the wrapper interace. On directions, f takes into account the current outputs of each of the constituent interfaces, as well as any new output given to the wrapper interface, then uses all of this to give input to each of the constituent interfaces. In particular, a judiciously chosen on-directions function could feed output from some interfaces as inputs to others. When f is a wrapper around a parallel product of interfaces, we call f the *interaction pattern* between those interfaces.

Example 3.47 (Repeater). Suppose we have a dependent dynamical system $\phi\colon Sy^S \to Ay + y$, which takes unchanging input and sometimes returns elements of A as output while other times returning only silence. What if we wanted to construct a system ψ that operates just like ϕ, but *always* returns elements of A as output? Where ϕ would have returned silence, we want ψ to instead *repeat* the last element of A that it returned. (We allow ψ to repeat an arbitrary element of A if ϕ returns silence before it has returned any elements of A yet.)

Let's think like a programmer. What we need is a way to store an element of A and then retrieve it. So whenever ϕ returns an element of A as output, we store it; then when ϕ returns silence, we retrieve the last element of A we stored and return that instead.

What should this storage-retrieval dynamical system look like? It needs to take elements of A as input, return elements of A as output, and store elements of A as states. In fact, the identity lens $\iota\colon Ay^A \to Ay^A$ works perfectly: it returns the element of A currently stored as output and updates its state to the input it receives.

Now we can juxtapose our original system ϕ with the storage-retrieval system ι by taking their parallel product, yielding a dynamical system

$$\phi \otimes \iota \colon SAy^{SA} \to (Ay + y) \otimes Ay^A$$

that runs both systems simultaneously—and independently. But what we want is for ϕ and ι to interact with each other, and for the resulting system to only output elements of A. To do so, we need to wrap an interface Ay around $\phi \otimes \iota$ by composing it with some lens

$$f \colon (Ay + y) \otimes Ay^A \to Ay,$$

the interaction pattern between the interfaces $Ay + y$ and Ay^A, which we must define.

Since $\otimes$ distributes over $+$, it suffices to give maps

$$g \colon Ay \otimes Ay^A \to Ay \qquad \text{and} \qquad h \colon y \otimes Ay^A \to Ay.$$

The former corresponds to the case where ϕ outputs an element of A, while the latter corresponds to the case where ϕ is silent.

When ϕ outputs an element of A, we want to return that output, but we also want to give that output as input to ι so that it can be stored. We don't need to do anything with the output of ι; we can simply discard it. So g should send $(a, a') \mapsto a$ on positions, returning the output of ϕ and discarding the output of ι; and the on-directions function $g^\sharp_{(a,a')} \colon 1 \to A$ should specify the direction $a \in A$, feeding the output of ϕ as input to ι.

Meanwhile, when ϕ outputs silence, we want to return the output of ι instead. We also need to feed the output of ι back into ι as input so that it can continue to be stored. So h should be the identity on positions as well as the identity on directions.

Example 3.48 (Paddling). Say we wanted to build a Moore machine with interface $\mathbb{N}y$; we may interpret its natural number output as the machine's current location. What if we don't want this machine to jump around wildly? Instead, suppose we want to be very strict about what how far the machine can move and what makes it move.

To accomplish this, we introduce two intermediary systems, which we call the *paddler* and the *tracker*:[a]

$$\text{paddler} \colon Sy^S \to 2y \qquad \text{and} \qquad \text{tracker} \colon Ty^T \to \mathbb{N}y^2$$

The paddler has interface $2y$ because it is blind (i.e. takes no inputs) and can only move (i.e. output) its paddle to the left side or the right side: $2 \cong \{\text{left}, \text{right}\}$. The tracker has interface $\mathbb{N}y^2$ because it will announce the location of the machine (as an element $n \in \mathbb{N}$) and watch what side the paddler is on (as an element of 2). We can wrap an interface $\mathbb{N}y$ around them both using an interaction pattern

$$2y \otimes \mathbb{N}y^2 \to \mathbb{N}y$$

whose on-positions function is the projection $2\mathbb{N} \to \mathbb{N}$, returning the location returned by the tracker, and whose on-directions function is the projection $2\mathbb{N} \to 2$, passing the output of the paddler as input to the tracker.

Let's leave the paddler's dynamics alone—how you make that paddler behave is totally up to you—and instead focus on the dynamics of the tracker. We want it to watch for when the paddle switches from left to right or from right to left; at that moment it should push the paddler forward one unit. Thus the states of the tracker are given by $T := 2\mathbb{N}$, storing what side the paddler is on and the current location. The on-positions function of the tracker is the projection $2\mathbb{N} \to \mathbb{N}$ that returns the current location; then at each $(d, i) \in 2\mathbb{N}$, the on-directions function of the tracker $2 \to 2\mathbb{N}$ sends

$$d' \mapsto \begin{cases} (d', i) & \text{if } d = d' \\ (d', i+1) & \text{if } d \neq d', \end{cases}$$

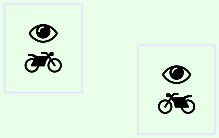
storing the new direction of the paddler as well as moving the machine forward one unit if the paddle switches while keeping the machine still if the paddle stays still.

[a] Perhaps one could refer to the tracker as the *demiurge*; it is responsible for maintaining the material universe.

Exercise 3.49 (Solution here). Change the dynamics and state system of the tracker in Example 3.48 so that it exhibits the following behavior.

When the paddle switches once and stops, the tracker increases its location by one unit and stops, as before in Example 3.48. But when the paddle switches twice in a row, the tracker increases its location by two units on the second switch! So if it is quiet for a while and then switches three times in a row, the tracker will increase its location by one then two then two. ◊

Example 3.50. Suppose you have two systems with the same interface $p := q := \mathbb{R}^2 y^{\mathbb{R}^2 - \{(0,0)\}}$.



The output of each interface indicates the location of the system, while the range of possible inputs indicate the locations that the system could observe, relative to the location of the system itself. Taking all pairs of reals except $(0, 0)$ corresponds to the fact that the eye cannot see that which is at the same position as the eye.

Let's have the two systems constantly approaching each other with a force equal to the reciprocal of the squared distance between them. If they finally collide, let's have the whole thing come to a halt. To do this, we want the wrapper interface to be $\{\text{`go'}\}y + \{\text{`stop'}\}$, so that if the system returns 'go' it can still advance to the next state, but if it returns 'stop' it halts. The wrapper $\mathbb{R}^2 y^{\mathbb{R}^2 - \{(0,0)\}} \otimes \mathbb{R}^2 y^{\mathbb{R}^2 - \{(0,0)\}} \to \{\text{`go'}\}y + \{\text{`stop'}\}$ is given on positions by

$$\big((x_1, y_1), (x_2, y_2)\big) \mapsto \begin{cases} \text{`stop'} & \text{if } x_1 = x_2 \text{ and } y_1 = y_2 \\ \text{`go'} & \text{otherwise.} \end{cases}$$

On directions, we use the function

$$\big((x_1, y_1), (x_2, y_2)\big) \mapsto \big((x_2 - x_1, y_2 - y_1), (x_1 - x_2, y_1 - y_2)\big),$$

so that each system is able to see the location of the other system relative to its own, i.e. the vector pointing from itself to the other system (unless that vector is zero, in which case the whole thing should have already halted).

We can use these vectors to define the internal dynamics of each system so that they move the way we want them to. Each system will hold as its internal state its current location and velocity, i.e. $S = \mathbb{R}^2 \times \mathbb{R}^2$. To define a lens $Sy^S \to \mathbb{R}^2 y^{\mathbb{R}^2 - \{(0,0)\}}$ we simply return the current location, update the current location by adding the current velocity, and update the current velocity by adding a vector with appropriate magnitude pointing to the other system:

$$\begin{aligned} \mathbb{R}^2 \times \mathbb{R}^2 &\xrightarrow{\text{return}} \mathbb{R}^2 \\ \big((x, y), (v_x, v_y)\big) &\xmapsto{\text{return}} (x, y) \end{aligned}$$

$$\begin{aligned} \mathbb{R}^2 \times \mathbb{R}^2 \times (\mathbb{R}^2 - \{(0,0)\}) &\xrightarrow{\text{update}} \mathbb{R}^2 \times \mathbb{R}^2 \\ \big((x, y), (v_x, v_y), (a, b)\big) &\xmapsto{\text{update}} \left(x + v_x, y + v_y, v_x + \frac{a}{(a^2 + b^2)^{3/2}}, v_y + \frac{b}{(a^2 + b^2)^{3/2}}\right) \end{aligned}$$

Exercise 3.51 (Solution here). Suppose (X, d) is a metric space, i.e. X is a set of points and $d \colon X \times X \to \mathbb{R}_{\geq 0}$ is a distance function satisfying the usual laws. Let's have robots interact in this space.

Let A, A' be sets, each thought of as a set of signals, and let $a_0 \in A$ and $a'_0 \in A'$ be elements, each thought of as a default value. Let $p := AXy^{A'X}$ and $p' := A'Xy^{AX}$, and imagine there are two robots, one with interface p, returning a signal as an element of A and its location as a point in X, and one with interface p', returning a signal as an element of A' and also its location as a point in X.

1. Write down an interaction pattern $p \otimes p' \to y$ such that each robot receives the

other's location, but that it only receives the other's signal when their locations x, x' are sufficiently close, namely when $d(x, x') < 1$. Otherwise, it receives the default signal.

2. Write down an interaction pattern $p \otimes p' \to y^{[0,5]}$ where the value $s \in [0, 5]$ is a scalar, allowing the signal to travel s times further.
3. Suppose that each robot has a set S, S' of possible private states in addition to their locations. What functions are involved in providing a dynamical system $\phi\colon SXy^{SX} \to AXy^{A'X}$, if the location state $x \in X$ is directly returned without modification?
4. Change the setup in any way so that each robot only extends a port to hear the other's signal when the distance between them is less than s. Otherwise, they can only detect the position (element of X) that the other currently inhabits. (Don't worry too much about timing—one missed signal when the robots first get close or one extra signal when the robots first get far is okay.) ◊

3.4.2 Enclosing juxtaposed dynamical systems together

We saw in Section 3.3.4 that a situation (i.e. lens into y) for the interface of a dynamical system encloses that dynamical system in a closed system. So it should not come as a surprise that a situation for a parallel product of interfaces yields an interaction pattern between the interfaces that only allows the interfaces to interact with each other, cutting off any other interaction with the outside world.

Example 3.52 (Picking up the chalk). Imagine that you see some chalk and you pinch it between your thumb and forefinger. An amazing thing about reality is that you will then have the chalk, in the sense that you can move it around. How might we model this in **Poly**? We will construct a closed dynamical system—one with interface y—consisting of only you and the chalk. To do so, we will provide an interface for you, and interface for the chalk, and an enclosure for your juxtaposition.

Let's say that your hand can be at one of two heights, down or up, and that you can either press (apply pressure between your thumb and forefinger) or not press. Let's also say that you take in information about the chalk's height. Here are the two sets we'll be using:

$$H := \{\text{'down'}, \text{'up'}\} \qquad \text{and} \qquad P := \{\text{'press'}, \text{'no press'}\}.$$

Your interface is HPy^H: returning your own height and pressure, and receiving the chalk's height.

As for the chalk, it is either 'in' your possession or 'out' of it. Either way, it also returns its height, which is either 'down' or 'up' in the air. The chalk always takes in information about whether pressure is being applied or not. When it's 'out' of your possession, that's the whole story, but when it is 'in' your possession, it also receives your hand's height. All together, here are the two interfaces:

$$\texttt{You} \coloneqq HPy^H \qquad \text{and} \qquad \texttt{Chalk} \coloneqq \{\text{`out'}\}Hy^P + \{\text{`in'}\}Hy^{HP}.$$

Now we want to give the interaction pattern between you and the chalk. As we said before, you see the chalk's height. If your hand is not at the height of the chalk, the chalk receives no pressure. Otherwise, your hand is at the height of the chalk, so the chalk receives your pressure (or lack thereof). Furthermore, if the chalk is in your possession, it also receives your hand's height.

To provide a map $\gamma\colon \texttt{You} \otimes \texttt{Chalk} \to y$, we use the fact that $\texttt{Chalk}$ is a sum and that $\otimes$ distributes over $+$. Thus we need to give two maps

$$\alpha\colon HPy^H \otimes Hy^P \to y \qquad \text{and} \qquad \beta\colon HPy^H \otimes Hy^{HP} \to y$$

The map β, corresponding to when the chalk is in your possession, is quite easy to describe; it can be unfolded to a function $HPH \to HHP$, and we take it to be the obvious map sending your height and pressure to the chalk and the chalk's height to you; see Exercise 3.54. But α is more semantically interesting: it is given by the map

$$(h_{\texttt{You}}, p_{\texttt{You}}, h_{\texttt{Chalk}}) \mapsto \begin{cases} (h_{\texttt{Chalk}}, \text{`no press'}) & \text{if } h_{\texttt{You}} \neq h_{\texttt{Chalk}} \\ (h_{\texttt{Chalk}}, p_{\texttt{You}}) & \text{if } h_{\texttt{You}} = h_{\texttt{Chalk}}. \end{cases}$$

So now we've got you and the chalk in enclosed together by γ, so we are ready to add some dynamics. Your dynamics can be whatever you want, so let's just add some dynamics to the chalk (you'll get to give yourself some dynamics in Exercise 3.54). The chalk has only four states $C \coloneqq \{\text{`out'}, \text{`in'}\} \times H \cong 4$: the H coordinate is its current height, and the other coordinate is whether or not it is in your possession. We will give a dynamical system $Cy^C \to \texttt{Chalk}$ with states C and interface $\texttt{Chalk}$, i.e. a lens

$$\{\text{`out'}, \text{`in'}\} \times Hy^{\{\text{`out'}, \text{`in'}\} \times H} \to \{\text{`out'}\}Hy^P + \{\text{`in'}\}Hy^{HP}. \tag{3.53}$$

On positions, as you might guess, the chalk returns its height and whether it is in your possession directly. On directions, if it's not in your possession, it falls down unless you catch it (i.e. apply pressure to it so that it enters your possession); if it is in your possession, it takes whatever height you give it. So we can express the on-directions function of Eq. (3.53) at ($\text{`out'}, h_{\texttt{Chalk}}$) as

$$\begin{aligned} \text{`no press'} &\mapsto (\text{`out'}, \text{`down'}) \\ \text{`press'} &\mapsto (\text{`in'}, h_{\texttt{Chalk}}) \end{aligned}$$

and the on-directions function of (3.53) at ($\text{`in'}, h_{\texttt{Chalk}}$) as

$$(h_{\texttt{You}}, \text{`no press'}) \mapsto (\text{`out'}, h_{\texttt{You}})$$

$$(h_{\texttt{You}}, \text{'press'}) \mapsto (\text{'in'}, h_{\texttt{You}}).$$

Obviously, this is all quite complicated, intricate, and contrived. Our goal here is to show that you can define interactions in which one system can engage with or disengage from another, where one system controls the behavior of the other when the two are engaged.

Exercise 3.54 (Solution here).

1. In Example 3.52, we said that $\beta\colon HPy^{H} \otimes Hy^{HP} \to y$ was easy to describe and given by a function $HPH \to HHP$. Explain what's being said, and provide the function.
2. Provide dynamics to the `You` interface (i.e. specify a dynamical system with interface `You`) so that you repeatedly reach down and grab the chalk, lift it with your hand, and drop it. ◊

Given $n \in \mathbb{N}$ and polynomials $p_1, \ldots, p_n$ as interfaces, a situation $p_1 \otimes \cdots \otimes p_n \to y$ puts these n interfaces in an enclosure together. The following proposition provides an alternative perspective on such situations.

Proposition 3.55. Given polynomials $p, q \in \mathbf{Poly}$, there is a bijection

$$\Gamma(p \otimes q) \cong \mathbf{Set}\big(q(1), \Gamma(p)\big) \times \mathbf{Set}\big(p(1), \Gamma(q)\big). \tag{3.56}$$

The idea is that specifying an enclosure for interfaces p and q together is equivalent to specifying an enclosure for p for every output q might return and specifying an enclosure for q for every output p might return.

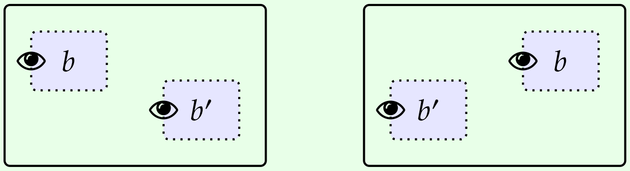
Proof of Proposition 3.55. This is a direct calculation:

$$\begin{aligned}\Gamma(p \otimes q) &\cong \prod_{i \in p(1)} \prod_{j \in q(1)} (p[i] \times q[j]) \\ &\cong \left(\prod_{j \in q(1)} \prod_{i \in p(1)} p[i]\right) \times \left(\prod_{i \in p(1)} \prod_{j \in q(1)} q[j]\right) \\ &\cong \mathbf{Set}(q(1), \Gamma(p)) \times \mathbf{Set}(p(1), \Gamma(q)).\end{aligned}$$

□

Example 3.57. An enclosure $f\colon By^{A} \otimes B'y^{A'} \to y$, corresponds to a map $BB' \to AA'$. In other words, for every pair of outputs $(b, b') \in BB'$, the enclosure f specifies a pair of inputs $(a, a') \in AA'$.

Let's think of elements of B and B' not as outputs, but as locations that two machines may occupy.



Then given a pair of locations (b, b'), the interaction pattern f tells us what the two eyes see, i.e. what values of (a, a') they get. Equivalently, Eq. (3.56) says that the interaction pattern tells us what values the first eye sees at any location when the second eye's location is fixed at b', as well as what values the second eye sees at any location when the first eye's location is fixed at b.

Here we see that (3.56) provides two ways to interpret the interaction pattern between two interfaces in a closed system: either as an enclosure around each interface that the other is part of, or as a single enclosure around them both.

Exercise 3.58 (Solution here). Let $p \coloneqq q \coloneqq \mathbb{N}y^{\mathbb{N}}$. We wish to specify an enclosure around their juxtaposition.

1. Say we wanted to feed the output of q as input to p. What function $f\colon q(1) \to \Gamma(p)$ captures this behavior?
2. Say we wanted to feed the sum of the outputs of p and q as input to q. What function $g\colon p(1) \to \Gamma(q)$ captures this behavior?
3. What enclosure $\gamma\colon p \otimes q \to y$ does the pair of functions (f, g) correspond to via (3.56)?
4. Let dynamical systems $\phi\colon \mathbb{N}y^{\mathbb{N}} \to p$ and $\psi\colon \mathbb{N}y^{\mathbb{N}} \to q$ both be the identity on $\mathbb{N}y^{\mathbb{N}}$. Suppose ϕ starts in the state $0 \in \mathbb{N}$ and ψ starts in the state $1 \in \mathbb{N}$. Describe the behavior of the system obtained by enclosing ϕ and ψ together with γ, i.e. the system $(\phi \otimes \psi) \mathbin{⨟} \gamma$. ◊

Exercise 3.59 (Solution here). We will use (3.56) to consider the interaction pattern γ between `You` and `Chalk` from Example 3.52 as a pair of functions $\texttt{You}(1) \to \Gamma(\texttt{Chalk})$ and $\texttt{Chalk}(1) \to \Gamma(\texttt{You})$.

1. How does the chalk's output specify an enclosure for you? That is, write the map $\texttt{Chalk}(1) \to \Gamma(\texttt{You})$.
2. How does your output specify an enclosure for the chalk? That is, write the map $\texttt{You}(1) \to \Gamma(\texttt{Chalk})$. ◊

Exercise 3.60 (Solution here).

1. State and prove a generalization of (3.56) from Proposition 3.55 for n-many polynomials $p_1, \ldots, p_n \in \textbf{Poly}$.

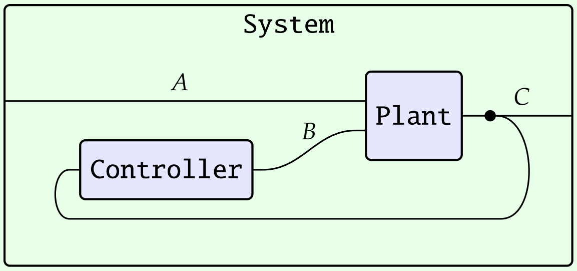
2. Generalize the "idea" statement between Proposition 3.55 and its proof. ◊

3.4.3 Wiring diagrams as interaction patterns

We first saw interactions between systems drawn as wiring diagrams in Section 1.2. They depict systems as boxes, showing how they can send inputs and outputs to each other through the wires between them, as well as how multiple systems can combine to form a larger system whenever smaller boxes are nested within a larger box.

Formally, and more precisely, we can think of each box in a wiring diagram as an interface given by some monomial. Then the entire wiring diagram—specifying how these boxes nest within a larger box—is just an interaction pattern between the interfaces, with the larger box playing the role of the wrapper interface.

Example 3.61. Here is a simple wiring diagram.



(3.62)

The plant is receiving information from the world outside the system, as well as from the controller. It's also producing information for the outside world which is being monitored by the controller.

There are three boxes shown in (3.62): the controller, the plant, and the system. Each has a fixed set of inputs and outputs, and so we can consider the box as a monomial interface.

$$\texttt{Plant} \coloneqq Cy^{AB} \qquad \texttt{Controller} \coloneqq By^{C} \qquad \texttt{System} \coloneqq Cy^{A}. \tag{3.63}$$

The wiring diagram itself is a wrapper

$$w \colon \texttt{Plant} \otimes \texttt{Controller} \to \texttt{System},$$

specifying an interaction pattern between `Plant` and `Controller` with wrapper interface `System`. Concretely, w is a lens $CBy^{ABC} \to Cy^{A}$ that dictates how wires are feeding from outputs to inputs. Like all lenses between monomials, w consists of an on-positions function $CB \to C$ and an on-directions function $CBA \to ABC$.

The wiring diagram is a picture that tells us which maps to use. The on-positions function says "inside the system you have boxes outputting values of type C and B. The system needs to produce an output of type C; how shall I obtain it?" The answer, according to the wiring diagram, is to send $(c, b) \mapsto c$.

Meanwhile, the on-directions function says "inside the system you have boxes outputting values of type C and B, and the system itself is receiving an input value of type A. The boxes inside need input values of type A, B, and C; how shall I obtain them?" Again, we can read the answer off the wiring diagram: send $(c, b, a) \mapsto (a, b, c)$.

Note that neither the wiring diagram nor any of the boxes within it are dynamical systems on their own. Rather, each box is a monomial that could be the interface of a dynamical system. When we assign to a box a dynamical system having that box as its interface, we say that *give dynamics* to the box. So the entire wiring diagram is a wrapper that tells us how, given the dynamics for each inner box,

$$\phi\colon Sy^S \to \texttt{Plant} \qquad \text{and} \qquad \psi\colon Ty^T \to \texttt{Controller},$$

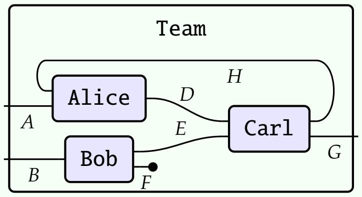
we can obtain the dynamics for the outer box:

$$STy^{ST} \xrightarrow{\phi \otimes \psi} \texttt{Plant} \otimes \texttt{Controller} \xrightarrow{w} \texttt{System}.$$

Exercise 3.64 (Solution here).

1. Make a new wiring diagram like (3.62) except where the controller also receives information of type A' from the outside world.
2. What are the monomials represented by the boxes in your diagram (replacing (3.63))?
3. What is the interaction pattern represented by this wiring diagram? Give the corresponding lens, including its on-positions and on-directions functions. ◊

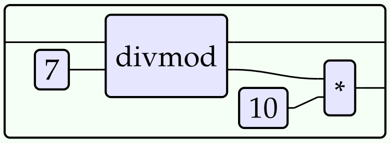
Exercise 3.65 (Solution here). Consider the following wiring diagram.



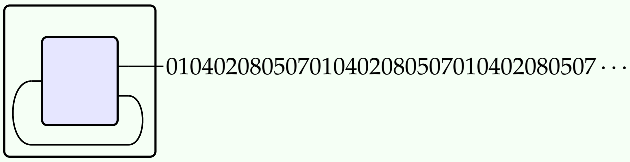
1. Write out the polynomials for each of `Alice`, `Bob`, and `Carl`.
2. Write out the polynomial for the outer box, `Team`.
3. The wiring diagram constitutes a lens f in **Poly**; what is its domain and codomain?
4. What lens is it?
5. Suppose we have dynamical systems $\alpha\colon Ay^A \to \texttt{Alice}$, $\beta\colon By^B \to \texttt{Bob}$, and $\gamma\colon Cy^C \to \texttt{Carl}$. What is the induced dynamical system with interface `Team`? ◊

Exercise 3.66 (Long division; solution here).

1. Let divmod: $\mathbb{N} \times \mathbb{N}_{\geq 1} \to \mathbb{N} \times \mathbb{N}$ send $(a, b) \mapsto (a \text{ div } b, a \bmod b)$; for example, it sends $(10, 7) \mapsto (1, 3)$ and $(30, 7) \mapsto (4, 2)$. Use Exercise 3.9 to turn it into a dynamical system.
2. Interpret the following wiring diagram, where we have already given dynamics to each box as indicated by their labels:



3. Use the above and a diagram of the following form to create a dynamical system that alternates between spitting out 0's and the base-10 digits of 1/7 after the decimal point, like so:



We will see in **??** how to use a system to build another dynamical system that only returns every other output of the original system, then apply this to the above system in **??**.

◊

Example 3.67 (Graphs as wiring diagrams and cellular automata). Suppose we have a graph $G = (E \rightrightarrows V)$ as in Definition 3.23 and a set $\tau(v)$ associated with each vertex $v \in V$:

$$\begin{tikzcd} E \arrow[r, shift left, "s"] \arrow[r, shift right, "t"'] & V \arrow[r, "\tau"] & \mathbf{Set} \end{tikzcd}$$

We can think of G as an alternative representation of a specific kind of wiring diagram, one where each inner box has exactly one output wire and the outer box is closed. The vertices $v \in V$ are the inner boxes, the set $\tau(v)$ is the set associated with v's output wire, and each edge e is a wire connecting the output wire of its target $t(e)$ to an input wire of its source $t(e)$. An edge from a vertex v_0 to a vertex v_1 indicates that the inputs to v_0 depend on the outputs of v_1.[a]

In other words, we can associate each vertex $v \in V$ with the monomial

$$p_v \coloneqq \tau(v) y^{\prod_{e \in E_v} \tau(t(e))}$$

specifying its inputs and outputs, where $E_v \coloneqq s^{-1}(v) \subseteq E$ denotes the set of edges emanating from v. The graph then determines an enclosure

$$\gamma\colon \bigotimes_{v \in V} p_v \to y$$

given by a function

$$\prod_{v \in V} \tau(v) \longrightarrow \prod_{e \in E} \tau(t(e))$$

that sends each dependent function $o\colon (v \in V) \to \tau(v)$ to the dependent function $(e \in E) \to \tau(t(e))$ sending $e \mapsto o(t(e))$. In other words, given the output $o(v) \in \tau(v)$ for every vertex $v \in V$, we know for each edge $e \in E$ that the input $s(e)$ receives is the output of $t(e)$.

Hence, once we give dynamics to each p_v, namely by specifying a dynamical system $S_v y^{S_v} \to p_v$ with outputs in $\tau(v)$ and inputs in $\prod_{e \in E_v} \tau(t(e))$, we will obtain a closed dynamical system that transitions from each vertex's state to the next according to the information that they pass each other along their edges.

Effectively, by interpreting a graph as a wiring diagram and giving each vertex dynamics, we have created what is known as a *cellular automaton*—a network of vertices (or *cells*) with states, such that each vertex $v \in V$ "listens" to the signals its *neighbors* in E_v send based on their states, then responds accordingly by updating its own state.

For example, a common graph found in cellular automata is a 2-dimensional integer lattice, with vertices $V := \mathbb{Z} \times \mathbb{Z}$. The edges indicate which vertices are neighbors and thus "hear" which other vertices. One might use

$$E := (\{-1, 0, 1\} \times \{-1, 0, 1\} - \{(0, 0)\}) \times V$$

with $s(i, j, m, n) = (m, n)$ and $t(i, j, m, n) = (m + i, n + j)$, so that the neighbors of each vertex are the eight vertices that surround it.

[a] We could have defined the edges in the opposite directions, so that they would point in the direction of data flow rather than in the direction of data dependencies; this was an arbitrary choice.

Exercise 3.68 (Conway's Game of Life; solution here). Conway's Game of Life is played on a 2-dimensional integer lattice as follows. Each lattice point is either *live* or *dead*, and each point observes its eight *neighbors* to which it is horizontally, vertically, or diagonally adjacent. The following occurs at every time step:

- Any live point with 2 or 3 live neighbors remains live.
- Any dead point with 3 live neighbors becomes live.
- All other points either become or remain dead.

We can use Example 3.67 to model Conway's Game of Life as a closed dynamical system.

1. What is the appropriate graph $E \rightrightarrows V$?
2. What is the appropriate assignment of sets $\tau\colon V \to \mathbf{Set}$?

3. What are the polynomials p_v from Example 3.67?
4. What is the appropriate state set S_v for each interface p_v?
5. What is the appropriate dynamical system map $S_v y^{S_v} \to p_v$? ◊

3.4.4 More examples of general interaction

While wiring diagrams are a handy visualization tool for certain simple interaction patterns, there are more general interaction patterns that cannot be captured by such a diagram. For example, here we generalize our previous cellular automata example.

Example 3.69 (Generalized cellular automata: voting on who your neighbors are). Recall from Example 3.67 how we constructed a cellular automaton on a graph $G = (E \rightrightarrows V)$. For each $v \in V$, the graph specifies the set $N(v) := t(E_v)$ of vertices at the ends of edges coming out of v. These vertices are the *neighbors* of v, or the vertices that v can "listen" to. We call the map $N\colon V \to \mathsf{2}^V$ from each vertex to the set of its neighbors the *neighbor function*. For simplicity, we let each vertex store and return one of two states, so $S_v := \tau(v) := \mathsf{2}$.

Now just take the vertices and forget the edges of our graph. Suppose instead that we are given a function $n\colon V \to \mathbb{N}$ that we think of as specifying the number $n(v)$ of neighbors each $v \in V$ could potentially have. Let $\mathsf{n}(v) := \{1, 2, \ldots, n(v)\}$. Then we can think of the monomial that each vertex represents as

$$p_v \cong \mathsf{2}y^{\mathsf{2}^{\mathsf{n}(v)}},$$

returning its own state as output and receiving its potential neighbors' states as input.

Say that a neighbor function $N\colon V \to \mathsf{2}^V$ *respects* n if we have an isomorphism $N(v) \cong \mathsf{n}(v)$ for each $v \in V$. Now suppose we have a function $N'_{-}\colon \mathsf{2}^V \to (\mathsf{2}^V)^V$ that sends each set of vertices $S \in \mathsf{2}^V$ to a neighbor function $N'_S\colon V \to \mathsf{2}^V$ that respects n. In other words, each possible state configuration S of all the vertices in V determines a neighbor function N'_S. In the case of Example 3.67, when we had a graph, it told us what the neighbor function should always be. Now we can think of it like all the vertices are voting, via N', on what neighbor function to use to determine which vertices are listening to which others.

We can put this all together by providing an enclosure for all the vertices,

$$\bigotimes_{v \in V} p_v \cong \mathsf{2}^V y^{\mathsf{2}^{\sum_{v \in V} \mathsf{n}(v)}} \longrightarrow y. \tag{3.70}$$

Specifying such an enclosure amounts to specifying a function $g\colon \mathsf{2}^V \to \mathsf{2}^{\sum_{v \in V} \mathsf{n}(v)}$ that assigns each possible state configuration $S \in \mathsf{2}^V$ of all the vertices in V to a function $g(S)\colon \sum_{v \in V} \mathsf{n}(v) \to \mathsf{2}$ specifying the states every vertex hears. But we already have a neighbor function assigned to S that respects n, namely N'_S, for which $N'_S(v) \cong \mathsf{n}(v)$ for all $v \in V$. So we can think of $g(S)$ equivalently as a function $g(S)\colon \sum_{v \in V} N'_S(v) \to \mathsf{2}$

that says for each $v \in V$ what signal in $\mathsf{2}$ it should receive from its neighbor $w \in N'_S(v)$. But we can just have it receive the current state of its neighbor, as given by S:

$$g(S)(v, w) \coloneqq S(w).$$

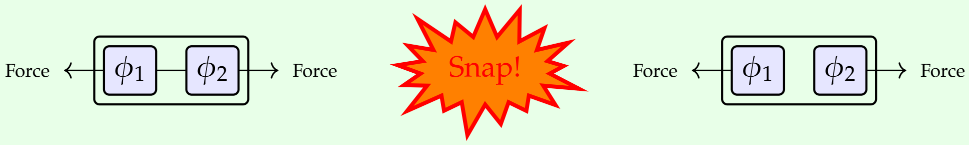
We have accomplished our goal: the vertices "vote" on how they ought to be connected, in that their states together determine the neighbor function. Of course, we don't mean to imply that this vote needs to be democratic or fair in any way: it is an arbitrary function $N'_{-}\colon \mathsf{2}^V \to (\mathsf{2}^V)^V$. It could be dictated by a given vertex $v_0 \in V$ in the sense that its state completely determines the neighbor function $V \to \mathsf{2}^V$; this would be expressed by saying that N'_{-} factors as $\mathsf{2}^V \to \mathsf{2}^{\{v_0\}} \cong \mathsf{2} \xrightarrow{I_0} (\mathsf{2}^V)^V$ for some I_0.

Exercise 3.71 (Solution here). We can change Example 3.69 slightly by replacing the wrapper interface y with some other interface.

1. First change it to Ay for some set A of your choice, and update (3.70) so that the system outputs some aspect of the current state configuration of all the vertices $S \in \mathsf{2}^V$.
2. What would it mean to change (3.70) to a map $\bigotimes_{v \in V} p_v \to y^A$ for some A? ◊

Finally, we are ready to formalize the examples we previewed in Section 1.2.

Example 3.72. Recall the first picture from Example 1.10. We said that when too much force is applied to a material, bonds can break. Let's simplify the picture a bit.



We will imagine the dependent dynamical systems $\phi_1\colon Sy^S \to p_1$ and $\phi_2\colon Sy^S \to p_2$ as initially connected in space. They experience forces from the outside world, and—for as long as they are connected—they experience forces from each other. More precisely, each interface is defined by

$$p_1 \coloneqq p_2 \coloneqq Fy^{FF} + \{\text{'snapped'}\}y^F.$$

Elements of F will be called *forces*. We need to be able to add and compare forces, i.e. we need F to be an ordered monoid; let's say $F = \mathbb{N}$ for simplicity. The idea is that the interface has two kinds of output it can return: either a force $f_i \in F$ on the other system, at which point it can receive an input in FF indicating a force acting on the system from its left and another force acting on it from its right; or 'snapped,' indicating that the system is no longer connected to the other system, at which point it only receives a single force in F from the outside.

The wrapper interface is defined to be

$$p \coloneqq y^{FF};$$

it takes as input two forces (f_L, f_R) and returns unchanging output.

Though the systems ϕ_1 and ϕ_2 may be initially connected, if the forces on either one surpass a threshold, that system stops sending and receiving forces from the other. The connection is broken and neither system ever receives forces from the other again. This is what we will implement explicitly below.

To do so, we need to define an interaction pattern $p_1 \otimes p_2 \to p$ that wraps p around ϕ_1 and ϕ_2. That is, we need to give a lens

$$\kappa\colon (Fy^{FF} + \{\text{‘snapped’}\}y^F) \otimes (Fy^{FF} + \{\text{‘snapped’}\}y^F) \to y^{FF}.$$

By the distributivity of $\otimes$ over $+$, it suffices to give four maps:

$$\begin{aligned}
\kappa_{11}\colon\ & FFy^{(FF)(FF)} && \to y^{FF}\\
\kappa_{12}\colon\ & F\{\text{‘snapped’}\}y^{(FF)F} && \to y^{FF}\\
\kappa_{21}\colon\ & \{\text{‘snapped’}\}Fy^{F(FF)} && \to y^{FF}\\
\kappa_{22}\colon\ & \{\text{‘snapped’}\}\{\text{‘snapped’}\}y^{FF} && \to y^{FF}
\end{aligned} \tag{3.73}$$

The middle two maps κ_{12} and κ_{21} won’t actually occur in our dynamics, so we take them to be arbitrary. We take the last map κ_{22} to be the obvious isomorphism, passing the forces from outside to the two internal interfaces. The first map κ_{11} is equivalent to a function $(FF)(FF) \to (FF)(FF)$ which we take to be $((f_1, f_2), (f_L, f_R)) \mapsto ((f_L, f_2), (f_1, f_R))$. While the multiple F’s may be a little hard to keep track of, what this map says is that if ϕ_1 returns the force f_1 on ϕ_2 as output and ϕ_2 returns the force f_2 on ϕ_1 as output, then ϕ_1 receives the force f_2 from the right as input and ϕ_2 receives the force f_1 from the left as input; and in the meantime the left external force f_L is given to ϕ_1 on the left, while the right external force is given to ϕ_2 on the right.

Now that we have the interfaces wrapped together, it remains to specify each dynamical system. The states in the two cases will be identical, namely $S \coloneqq F + \{‘snapped’\}$, meaning that at any point the system will either be in the state of applying a force to the other system or not. The dynamical systems themselves will be identical as well, up to a symmetry swapping left and right; let’s just define the left system for now. It is given by a lens

$$\phi_1\colon (F + \{\text{‘snapped’}\})y^{F+\{\text{‘snapped’}\}} \to Fy^{FF} + \{\text{‘snapped’}\}y^F$$

which we write as the sum of two maps

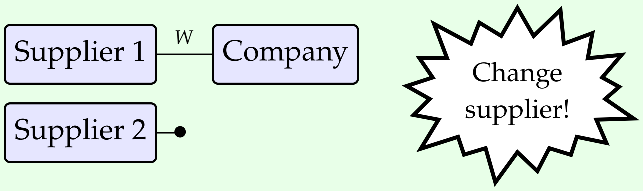
$$Fy^{F+\{\text{‘snapped’}\}} \to Fy^{FF} \qquad \text{and} \qquad \{\text{‘snapped’}\}y^{F+\{\text{‘snapped’}\}} \to \{\text{‘snapped’}\}y^{F}.$$

Both maps are the identity on positions, directly returning their current state. The second map corresponds to when the connection is broken, after which the connection should remain broken, so its on-directions function is constant, sending any input to ‘snapped.’ Meanwhile, the first map corresponds to the case where the systems are still connected; this system receives two input forces and must update its state—the force it applies—accordingly. We let the on-directions function $F(FF) \to F + \{\text{‘snapped’}\}$ send

$$(f_1, (f_L, f_2)) \mapsto \begin{cases} f_L & \text{if } f_1 + f_2 < 100 \\ \text{‘snapped’} & \text{otherwise} \end{cases}$$

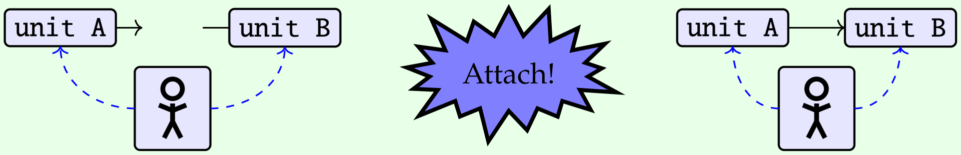
Thus, when the sum of forces is high enough, the internal state is updated to the ‘snapped’ state; otherwise, it is sent to the force it receives from outside, which it is now ready to transfer to the other system.

Example 3.74. Recall the second picture from Example 1.10. We want to consider the case of a company that may change its supplier based on its internal state. The company returns two possible outputs, corresponding to who it wants to receive widgets W from:



So the company has interface $2y^W$, and each supplier has interface Wy. Then an enclosure for the company and the suppliers is just a lens $2y^W \otimes Wy \otimes Wy \to y$, corresponding to a function $2W^2 \to W$ given by evaluation. In other words, the company's output determines its supplier.

Example 3.75. Recall the third picture from Example 1.10. When someone assembles a machine, their own outputs dictate the interaction pattern of the machine's components.



In order for the above picture to make sense, the output set of `unit A` should be the same as the input set of `unit B`. Call this set X, so that `unit A` has interface Xy and `unit B` has interface y^X. We fix a default value $x_0 \in X$ for the input to `unit B` when it is not connected to `unit A`. Meanwhile, the person takes no input and dictates whether the units are attached or not, so we give it interface $2y$.

Then an enclosure for the person and the units is a lens $2y \otimes Xy \otimes y^X \to y$. The morphism $2Xy^X \to y$, corresponding to a function $2X \to X$ that maps $(1, x) \mapsto x_0$ and $(2, x) \mapsto x$.

We can easily generalize Example 3.75. Indeed, we will see in the next section that there is an interface $[q_1 \otimes \cdots \otimes q_k\, ,\, r]$ that represents all the interaction patterns between $q_1, \ldots, q_k$ with wrapper interface r, and that wrapping it around p to it is just a larger interaction pattern:

$$\mathbf{Poly}(p, [q_1 \otimes \cdots \otimes q_k\, ,\, r]) \cong \mathbf{Poly}(p \otimes q_1 \otimes \cdots \otimes q_k\, ,\, r).$$

In other words, if p is deciding the interaction pattern between $q_1, \ldots, q_k$ with wrapper interface r, and gets feedback from that interaction pattern itself, then this is equivalent to an interaction pattern with wrapper interface r that p is part of alongside $q_1, \ldots, q_k$ inside of r.

What it also means is that if you want, you can put a little dynamical system inside of $[q_1 \otimes \cdots \otimes q_k, r]$ and have it be constantly choosing interaction patterns. Let's see how it works.

3.5 Closure of ⊗

The parallel monoidal product is closed—we have a monoidal closed structure on **Poly**—meaning that there is a closure operation, which we denote $[-,-]\colon \mathbf{Poly}^{\mathrm{op}} \times \mathbf{Poly} \to \mathbf{Poly}$, such that there is an isomorphism

$$\mathbf{Poly}(p \otimes q, r) \cong \mathbf{Poly}(p, [q, r]) \tag{3.76}$$

natural in p, q, r. The closure operation is defined on q, r as follows:

$$[q, r] \coloneqq \prod_{j \in q(1)} r \circ (q[j]y) \tag{3.77}$$

Here $\circ$ denotes standard functor composition; informally, $r \circ (q[j]y)$ is the polynomial you get when you replace each appearance of y in r by $q[j]y$. Composition, together with the unit y, is in fact yet another monoidal structure, as we will see in more depth in Part II.

Before we prove that the isomorphism (3.76) holds naturally, let us investigate the properties of the closure operation, starting with some simple examples.

Exercise 3.78 (Solution here). Calculate $[q, r]$ for $q, r \in \mathbf{Poly}$ given as follows.

1. $q \coloneqq 0$ and r arbitrary.
2. $q \coloneqq 1$ and r arbitrary.
3. $q \coloneqq y$ and r arbitrary.
4. $q \coloneqq A$ for $A \in \mathbf{Set}$ (constant) and r arbitrary.
5. $q \coloneqq Ay$ for $A \in \mathbf{Set}$ (linear) and r arbitrary.
6. $q \coloneqq y^2 + 2y$ and $r \coloneqq 2y^3 + 3$. ◊

Exercise 3.79 (Solution here). Show that for any polynomials p_1, p_2, q, we have an isomorphism

$$[p_1 + p_2, q] \cong [p_1, q] \times [p_2, q].$$

◊

Exercise 3.80 (Solution here). Show that there is an isomorphism

$$[q, r] \cong \sum_{f\colon q \to r} y^{\sum_{j \in q(1)} r[f_1(j)]} \tag{3.81}$$

where the sum is indexed by $f \in \mathbf{Poly}(q, r)$. ◊

Exercise 3.82 (Solution here). Verify that (2.67) holds.t ◊

Example 3.83. For any $A \in \mathbf{Set}$ we have

$$[y^A, y] \cong Ay \qquad \text{and} \qquad [Ay, y] \cong y^A.$$

More generally, for any polynomial $p \in \mathbf{Poly}$ we have

$$[p, y] \cong \Gamma(p) y^{p(1)}. \tag{3.84}$$

All these facts follow directly from (3.77).

Exercise 3.85 (Solution here). Verify the three facts above. ◊

Exercise 3.86 (Solution here). Show that for any $p \in \mathbf{Poly}$, if there is an isomorphism $[[p, y], y] \cong p$, then p is either linear Ay or representable y^A for some A. Hint: first show that p must be a monomial. ◊

Proposition 3.87. With $[-,-]$ as defined in (3.77), there is a natural isomorphism

$$\mathbf{Poly}(p \otimes q, r) \cong \mathbf{Poly}(p, [q, r]). \tag{3.88}$$

Proof. We have the following chain of natural isomorphisms:

$$\begin{aligned}
\mathbf{Poly}(p \otimes q, r) &\cong \mathbf{Poly}\Big(\sum_{i \in p(1)} \sum_{j \in q(1)} y^{p[i]q[j]}, r\Big) \\
&\cong \prod_{i \in p(1)} \prod_{j \in q(1)} \mathbf{Poly}(y^{p[i]q[j]}, r) && \text{(Universal property of coproducts)} \\
&\cong \prod_{i \in p(1)} \prod_{j \in q(1)} r(p[i]q[j]) && \text{(Yoneda lemma)} \\
&\cong \prod_{i \in p(1)} \prod_{j \in q(1)} \mathbf{Poly}(y^{p[i]}, r \circ (q[j]y)) && \text{(Yoneda lemma)} \\
&\cong \mathbf{Poly}\Big(\sum_{i \in p(1)} y^{p[i]}, \prod_{j \in q(1)} r \circ (q[j]y)\Big) \\
& && \text{(Universal property of (co)products)} \\
&\cong \mathbf{Poly}(p, [q, r]).
\end{aligned}$$

□

Exercise 3.89 (Solution here). Show that for any p, q we have an isomorphism of sets

$$\mathbf{Poly}(p, q) \cong [p, q](1).$$

Hint: you can either use the formula (3.77), or just use (3.88) with the Yoneda lemma and the fact that $y \otimes p \cong p$. ◊

The closure of $\otimes$ implies that for any $p, q \in \mathbf{Poly}$, there is a canonical *evaluation* map

$$\text{eval}\colon [p, q] \otimes p \longrightarrow q. \tag{3.90}$$

As in any closed monoidal category, such an evaluation map has the universal property that for any $r \in \mathbf{Poly}$ and map $f\colon p \otimes q \to r$, there is a unique lens $f'\colon p \to [q, r]$ such that the following diagram commutes:

$$\begin{tikzcd}
p \otimes q \arrow[r, "f' \otimes q"] \arrow[rr, bend right, "f"'] & {[q, r] \otimes q} \arrow[r, "\text{eval}"] & r
\end{tikzcd}$$

Exercise 3.91 (Solution here). Obtain the evaluation map $\text{eval}\colon [p,q]\otimes p \longrightarrow q$ from (3.90). ◊

Exercise 3.92 (Solution here).

1. For any set S, obtain the map $Sy^S \to y$ whose on-directions map is the identity on S using eval and Example 3.83.
2. Show that four maps in (3.73) from Example 3.72, written equivalently as

$$\begin{aligned}
\kappa_{11}\colon\ & Fy^{FF}\otimes Fy^{FF} \to y^F\otimes y^F\\
\kappa_{12}\colon\ & Fy^{FF}\otimes y^F \quad\ \to y^F\otimes y^F\\
\kappa_{21}\colon\ & y^F\otimes Fy^{FF} \quad\ \to y^F\otimes y^F\\
\kappa_{22}\colon\ & y^F\otimes y^F \qquad\ \to y^F\otimes y^F,
\end{aligned}\tag{3.93}$$

   can be obtained by taking the parallel product of identity maps and evaluation maps. ◊

Example 3.94 (Modeling your environment without knowing what it is). Let's imagine a robot whose interface is an arbitrary polynomial p. Let's imagine it is part of an interaction pattern

$$f\colon (q_1\otimes\cdots\otimes q_n)\otimes p \to r$$

with some other robots whose interfaces are $q_1,\ldots,q_n$; let $q \coloneqq (q_1\otimes\cdots\otimes q_n)$. The interaction pattern induces a morphism $f'\colon q\to[p,r]$ such that the original system f factors through the evaluation $[p,r]\otimes p\to r$.

In other words, $[p,r]$ holds within it all of the possible ways p can interact with other systems when they are all wrapped in r. For example, in the case of $r\coloneqq y$, note that $[p,y]\cong\prod_{i\in p(1)}p[i]y$. That is, for each p-position it produces a direction there, which is just what p needs as input in a closed system.

Now suppose we were to populate the interface p with dynamics, a map $Sy^S\to p$. One could aim to choose a set S along with an interesting map $g\colon S\to\mathbf{Poly}(p,r)$. Then each state s would include a guess $g(s)$ about the state of its environment. This is not the real environment q, but just the environment as it affects p, namely $[p,r]$. The robot's states model environmental conditions.

Example 3.95 (Chu &). Suppose we have polynomials $p_1,p_2,q_1,q_2,r\in\mathbf{Poly}$ and morphisms

$$\varphi_1\colon p_1\otimes q_1\to r \qquad\text{and}\qquad \varphi_2\colon p_2\otimes q_2\to r.$$

One might call these "r-Chu spaces." One operation you can do with these as Chu spaces is to return something denoted $\varphi_1 \& \varphi_2$, or "φ_1 *with* φ_2" of the following type:

$$\varphi_1 \& \varphi_2 \colon (p_1 \times p_2) \otimes (q_1 + q_2) \to r$$

Suppose we are given a position in p_1 and a position in p_2. Then given a position in either q_1 or q_2, one evaluates either φ_1 or φ_2 respectively to get a position in r; given a direction there, one returns the corresponding direction in q_1 or q_2 respectively, as well as a direction in $p_1 \times p_2$ which is either a direction in p_1 or in p_2.

This sounds complicated, but it can be done formally, once we have monoidal closure. We first rearrange both φ_1, φ_2 to be p-centric, using monoidal currying:

$$\psi_1 \colon p_1 \to [q_1, r] \qquad \text{and} \qquad \psi_2 \colon p_2 \to [q_2, r]$$

Now we multiply to get $\psi_1 \times \psi_2 \colon p_1 \times p_2 \to [q_1, r] \times [q_2, r]$. Then we apply Exercise 3.79 to see that $[q_1, r] \times [q_2, r] \cong [q_1 + q_2, r]$, and finally monoidal-uncurry to obtain $(p_1 \times p_2) \otimes (q_1 + q_2) \to r$ as desired.

3.6 Exercise solutions

Solution to Exercise 3.4.

We can write the Moore machine from Example 3.2 as a lens $Sy^S \to By^A$ between monomials as follows.

1. As $S = 3$, the state system is $Sy^S = 3y^3$.
2. As $B = \{b_1, b_2\}$ and $A = \{\text{orange}, \text{green}\}$, the interface is $By^A = \{b_1, b_2\}y^{\{\text{orange}, \text{green}\}}$.
3. The on-positions function "get" sends $1 \mapsto b_1, 2 \mapsto b_2$, and $3 \mapsto b_2$.
4. The on-directions function "set" sends

$$\begin{aligned} (1, \text{orange}) &\mapsto 1, & (1, \text{green}) &\mapsto 2, \\ (2, \text{orange}) &\mapsto 1, & (2, \text{green}) &\mapsto 3, \\ (3, \text{orange}) &\mapsto 1, & (3, \text{green}) &\mapsto 3. \end{aligned}$$

Solution to Exercise 3.7.

At any point in time, the Moore machine in Example 3.6 is located somewhere on the 2-dimensional plane, say at the coordinates $(x, y) \in \mathbb{R}^2$. This location is its current state. Whenever we ask the machine to produce output, it will tell us those coordinates, since $\text{return}(x, y) = (x, y)$. But if we give the machine input of the form (r, θ) for some distance $r \in [0, 1]$ and direction $\theta \in [0, 2\pi)$, the machine will move by that distance, in that direction, going from (x, y) to

$$\text{update}(x, y, r, \theta) = (x, y) + r(\cos\theta, \sin\theta)$$

(here we treat $\mathbb{R}^2$ as a vector space, so that $r(\cos\theta, \sin\theta)$ is a vector of length r in the direction of θ).

Solution to Exercise 3.9.

1. We seek an (A, B)-Moore machine $By^B \to By^A$ corresponding to the function $f \colon A \times B \to B$. We know that an (A, B)-Moore machine $By^B \to By^A$ consists of a return function $B \to B$ and an update function $B \times A \to B$. So we can simply let the return function be the identity on B and the update function be $B \times A \cong A \times B \xrightarrow{f} B$, i.e. the function f with its inputs swapped.
2. Generally, such a machine is not memoryless. Unlike in Example 3.8, the update function $B \times A \cong A \times B \xrightarrow{f} B$ does appear to depend on its first input, namely the previous state, which f takes as its second input.

However, if f factors through the projection $\pi_A\colon A \times B \to A$, i.e. if f can be written as a composite $A \times B \xrightarrow{\pi_A} A \xrightarrow{f'} B$ for some $f'\colon A \to B$, then the resulting machine *is* memoryless: it is the memoryless Moore machine from Example 3.8 corresponding to f'.

3. Given a function $g\colon A \to B$, we can compose it with the projection $\pi_A\colon A \times B \to A$ to obtain a function $A \times B \xrightarrow{\pi_A} A \xrightarrow{g} B$, which sends its first input through g and discards its second input. Then the corresponding (A, B)-Moore machine $By^B \to By^A$ has the identity function on B as its return function and the composite $B \times A \xrightarrow{\pi_A} A \xrightarrow{g} B$, which discards its first input and sends its second input through g, as its update function.

Solution to Exercise 3.10.

For each of the following constructs, we find $A, B \in \mathbf{Set}$ such that the construct can be identified with a morphism $Sy^S \to By^A$, i.e. a function return: $S \to B$ and a function update: $S \times A \to S$.

1. Given a discrete dynamical system with states S and transition funtion $n\colon S \to S$, we can set $A \coloneqq B \coloneqq 1$. Then return: $S \to 1$ is unique, while update: $S \times 1 \to S$ is given by $S \times 1 \cong S \xrightarrow{n} S$. The corresponding Moore machine has unchanging input (you could think of it as a button that says “advance to the next state”) and unchanging output (which effectively tells us nothing). So it is just a set of states, and a deterministic way to move from state to state.
We could have also set $A \coloneqq 0$ and $B \coloneqq S$, so that return $\coloneqq n$ and update: $S \times 0 \to S$ is unique, but this formulation is somewhat less satisfying: this is a Moore machine that never moves between its states, effectively functioning as a lookup table between whatever state the machine happens to be in and its output, which happens to refer to some state.
2. Given a magma consisting of a set S and a function $m\colon S \times S \to S$, we can set $A \coloneqq S$ and $B \coloneqq 1$. Then return: $S \to 1$ is unique, while update: $S \times S \to S$ is equal to m. The corresponding Moore machine produces unchanging output. It uses the binary operation m to combine the current state with the input—which also refers to a state—to obtain the new state.
Alternatively, we could have set the update function to be m with its inputs swapped. The difference here is that the new state is given by applying m with the input on the left and the current state on the right, rather than the other way around. If m is noncommutative, this would yield a different Moore machine.
We could have also set $A \coloneqq 0$ and $B \coloneqq S^S$, so that update: $S \times 0 \to S$ is unique, while currying m gives return, so that return(s) is the function $S \to S$ given by $s' \mapsto m(s, s')$. Alternatively, return(s) could be the function $s' \mapsto m(s', s)$. Either way, this is a Moore machine that never moves between its states, functioning as a lookup table between the machine's current state and the function m partially applied to that state on one side or the other.
3. Given a set S and a subset $S' \subseteq S$, we can set $A \coloneqq 0$ and $B \coloneqq 2$. Then update: $S \times 0 \to S$ is unique, while we define return: $S \to 2$ by
$$\text{return}(s) = \begin{cases} 1 & \text{if } s \in S' \\ 2 & \text{if } s \notin S', \end{cases}$$
so that S' can be recovered from the return function as its fiber over 1. The corresponing Moore machine never moves between its states, but gives one of two outputs indicating whether or not the current state is in the subset S'.

Solution to Exercise 3.11.

We modify the Moore machine from Example 3.6 as follows. The original Moore machine had states $\mathbb{R}^2$, so to add a health meter with values in $[0, 10]$, we take the cartesian product to obtain the new set of states $\mathbb{R}^2 \times [0, 10]$. The inputs and outputs are unchanged, so the Moore machine is a lens

$$\mathbb{R}^2 \times [0, 10]y^{\mathbb{R}^2 \times [0,10]} \to \mathbb{R}^2 y^{[0,1] \times [0,2\pi)}.$$

Its return function $\mathbb{R}^2 \times [0,10] \to \mathbb{R}^2$ is the canonical projection, as the machine only outputs its location in $\mathbb{R}^2$ and not its health; while its update function

$$\mathbb{R}^2 \times [0,10] \times [0,1] \times [0,2\pi) \to \mathbb{R}^2 \times [0,10]$$

sends (x, y, h, r, θ) to

$$(x + hr\cos\theta, y + hr\sin\theta, h'),$$

where $h' = h/2$ if the machine's new x-coordinate $x + hr\cos\theta < 0$ and $h' = h$ otherwise.

Solution to Exercise 3.12.

1. The tape of a Turing machine has states $V^{\mathbb{Z}} \times \mathbb{Z}$, outputs V, and inputs $V \times \{L, R\}$, so as a Moore machine, it is a lens
$$t \colon (V^{\mathbb{Z}} \times \mathbb{Z})y^{V^{\mathbb{Z}} \times \mathbb{Z}} \to Vy^{V \times \{L,R\}}.$$
2. The return function of t should output the value at the current cell of the tape. So return: $V^{\mathbb{Z}} \times \mathbb{Z} \to V$ is the evaluation map: it sends (f, c) with $f \colon \mathbb{Z} \to V$ and $c \in \mathbb{Z}$ to $f(c)$. Then the update function of t should write the input value of V in the current cell of the tape, then shift the cell number down or up one according to whether the second input value is L (left) or R (right). So
$$\text{update} \colon (V^{\mathbb{Z}} \times \mathbb{Z}) \times (V \times \{L, R\}) \to V^{\mathbb{Z}} \times \mathbb{Z}$$
sends old tape $f \colon \mathbb{Z} \to V$, old cell number $c \in \mathbb{Z}$, new value $v \in V$, and direction $D \in \{L, R\}$ to the new tape $f' \colon \mathbb{Z} \to V$ defined by
$$f'(n) := \begin{cases} v & \text{if } n = c \\ f(n) & \text{if } n \neq c \end{cases}$$
and the new cell number $c - 1$ if $D = L$ and $c + 1$ if $D = R$.

Solution to Exercise 3.13.

Given a file $f \colon \mathsf{n} \to \mathsf{ascii}$, we construct our file-reader as a Moore machine as follows. There are many options for what states the machine should record, but we will use pairs of values $(i, t) \in \mathsf{n}^2$, where i is the entry where the file-reader is currently located, while t is the entry where the file-reader should stop. But we will also include a "stopped" state to record when the file-reader has already stopped. So the Moore machine is a lens

$$(\mathsf{n}^2 + \{\text{“stopped”}\})y^{\mathsf{n}^2 + \{\text{“stopped”}\}} \to (\mathsf{ascii} + \{\text{“done”}\})y^{\{(s,t) \mid 1 \leq s \leq t \leq n\} + \{\text{“continue”}\}}.$$

If the file-reader's current state is "stopped," then the file-reader should output "done." Otherwise, the file-reader should output the character at the entry where the file-reader is currently located. So its return function $\mathsf{n}^2 + \{\text{“done”}\} \to \mathsf{ascii} + \{\text{“done”}\}$ sends (i, t) to $f(i)$ and "done" to "done." Meanwhile, the update function

$$(\mathsf{n}^2 + \{\text{“done”}\}) \times (\{(s, t) \mid 1 \leq s \leq t \leq n\} + \{\text{“continue”}\}) \to \mathsf{n}^2 + \{\text{“done”}\}$$

sends any state with input (s, t) to the state (s, t). On the other hand, if the input is "continue," an old state (i, t) is sent to the new state $(i + 1, t)$ if $i + 1 \leq t$ and "done" otherwise. Finally, if the old state is "done" and the input is "continue," the new state is still "done."

Solution to Exercise 3.21.

Here we examine the halting deterministic state automaton depicted in (3.22), with left state (and initial state) 1, right state 2, bottom state 3, and arrows orange and green.

1. The set of states is $S := \mathsf{3} = \{1, 2, 3\}$.
2. The set of input symbols is $A := \{\text{orange}, \text{green}\}$.
3. The automaton should halt at the accept states, so the accept states are exactly the states that have no arrows coming out of them—in this case, only state 3. States 1 and 2 are not accept states.

4. Let the corresponding lens be $\phi\colon Sy^S \to y^A + 1$, or $\phi\colon 3y^3 \to y^{\{\text{orange},\text{green}\}} + 1$. According to the previous part, ϕ has a return function $\phi_1\colon S \to 2$ sending states 1 and 2, as non-accept states, to 1, and sending state 3, as an accept state, to 2. Then the two nontrivial update functions behave as follows, according to the targets of the arrows in (3.22):
$$\begin{aligned} \phi_1^\sharp\colon &\quad \text{orange} \mapsto 2, \text{green} \mapsto 1 \\ \phi_2^\sharp\colon &\quad \text{orange} \mapsto 3, \text{green} \mapsto 1, \end{aligned}$$
while $\phi_3^\sharp\colon 0 \to S$ is trivial.
5. Some examples of words accepted by this automaton include the word (orange, orange), the word (orange, green, orange, orange), and the word (green, orange, green, green, green, orange, orange).
6. Some words are not accepted by the automaton because they lead you to a non-accept state (1 or 2); others are not accepted by the automaton because they lead you to an accept state (3) too early. Some examples of the former possibility include the words (green, green) and (orange, green, orange, green), while some examples of the latter possibility include the words (green, orange, orange, green) and (orange, orange, orange, orange, orange, orange, orange).

Solution to Exercise 3.25.

No matter what graph you chose, Example 3.24 tells us that if you were to draw out the labeled transition diagram of its associated dynamical system, you would just end up with a picture of your graph! The vertices of your graph are the states, and the edges of your graph are the possible transitions between them.

Solution to Exercise 3.27.

We give a file-searcher $\psi\colon Sy^S \to q$ that acts just like the file-reader $\phi\colon Sy^S \to p$ from Example 3.26, except that it only emits output $o \in \mathsf{ascii}$ when $c = 100$. In place of any other ascii character, we'll have it output _ instead. So its possible outputs should form the set
$$q(1) := \{\text{'ready'}, \text{'busy'}\} \times \{100, \_\}.$$

The direction-sets of q can be defined in the same way we defined the direction-sets of p: for each $c \in \{100, \_\}$, we have
$$q[(\text{'ready'}, c)] := S \qquad \text{and} \qquad q[(\text{'busy'}, c)] := 1.$$

Then we define the return function ψ_1 like ϕ_1, but first checking to see if the character at the current entry is 100: so for all $(s,t) \in S$,
$$\psi_1(s,t) = \begin{cases} (\text{'ready'}, 100) & \text{if } s = t \text{ and } f(s) = 100 \\ (\text{'ready'}, \_) & \text{if } s = t \text{ and } f(s) \neq 100 \\ (\text{'busy'}, 100) & \text{if } s \neq t \text{ and } f(s) = 100 \\ (\text{'busy'}, \_) & \text{otherwise} \end{cases}$$

Then the update functions of ψ behave just like those of ϕ. For each $(s,t) \in S$ for which $s = t$, we define the update function $\psi^\sharp_{(s,t)}\colon S \to S$ to be the identity on S. On the other hand, for each $(s,t) \in S$ for which $s \neq t$, we let the update function $\psi^\sharp_{(s,t)}\colon 1 \to S$ specify the element $(s+1, t) \in S$, thus shifting its current entry up by 1.

Solution to Exercise 3.29.

We modify the dynamical system from Example 3.28.

1. Previously, the set of inputs at each output $(i,j) \in \mathsf{n} \times \mathsf{n}$ of our interface p was $D_i \times D_j$. But now we also want to be able to give the robot a "reward value" $r \in \mathbb{R}$. So our new input set should be $D_i \times D_j \times \mathbb{R}$, making
$$p' := \sum_{(i,j) \in \mathsf{n} \times \mathsf{n}} y^{D_i \times D_j \times \mathbb{R}}.$$

2. Previously, a state was just a position in $\mathsf{n} \times \mathsf{n}$. But now we want to be able to record a list of reward values as well. Since each reward value is a real number, it suffices to define the state set to be $S' \coloneqq \mathsf{n} \times \mathsf{n} \times \mathsf{List}(\mathbb{R})$.
3. The former return function $\phi_1 \colon S \to p(1)$ was the identity on $\mathsf{n} \times \mathsf{n}$. The new return function ϕ'_1 should still just yield the robot's current grid position, but since it is now a function from $S' = \mathsf{n} \times \mathsf{n} \times \mathsf{List}(\mathbb{R})$, it should instead be the canonical projection $\phi'_1 \colon \mathsf{n} \times \mathsf{n} \times \mathsf{List}(\mathbb{R}) \to \mathsf{n} \times \mathsf{n}$.

   For each former state $(i, j) \in \mathsf{n} \times \mathsf{n}$, the former update function $\phi^\sharp_{(i,j)} \colon D_i \times D_j \to \mathsf{n} \times \mathsf{n}$ sent $(d, e) \mapsto (i + d, j + e)$. With an extra component $(r_1, \ldots, r_k) \in \mathsf{List}(\mathbb{R})$ of the state, the new update function $(\phi')^\sharp_{(i,j,(r_1,\ldots,r_k))} \colon D_i \times D_j \times \mathbb{R} \to \mathsf{n} \times \mathsf{n} \times \mathsf{List}(\mathbb{R})$ sends $(d, e, r) \mapsto (i + d, j + e, (r_1, \ldots, r_k, r))$, updating the list of rewards.

Solution to Exercise 3.30.

We are given $n \in \mathbb{N}$ and an (A_i, B_i)-Moore machine with state set S, i.e. a lens $Sy^S \to B_i y^{A_i}$, for each $i \in \mathsf{n}$. The universal property of products in **Poly** gives us a lens

$$Sy^S \to \prod_{i \in \mathsf{n}} B_i y^{A_i} \cong \left(\prod_{i \in \mathsf{n}} B_i \right) y^{\sum_{i \in \mathsf{n}} A_i},$$

which is a $(\sum_{i \in \mathsf{n}} A_i, \prod_{i \in \mathsf{n}} B_i)$-Moore machine with state set S (after all, the product of monomials is still a monomial).

Solution to Exercise 3.32.

Given a dynamical system $\phi \colon Sy^S \to p$, we seek a new dynamical system $\phi' \colon Sy^S \to p'$ that has the added option to provide no input at a step so that the state does not change. We can think of this as having two different interfaces acting on the same system: the original interface p of ϕ, and a new interface with only one possible input—namely the option to provide no input at all—that does not change the state. This latter interface also does not need to distinguish between its outputs; it should have just one possible output that says nothing. So the second interface we want acting on Sy^S is y.

If y were the only interface acting on the system, we would have a Moore machine $\epsilon \colon Sy^S \to y$ whose return function is the unique function $S \to 1$ and whose update function is the identity function on S, since the input never changes the system. Then p' is the product of the two interfaces p and y, while $\phi' \colon Sy^S \to p'$ is the unique map induced by $\phi \colon Sy^S \to p$ and $\epsilon \colon Sy^S \to y$. In particular, $p' \cong py \cong \sum_{i \in p(1)} y^{p[i]+1}$, while ϕ' consists of a return function $\phi'_1 \colon S \to p(1)$ that is the same as the return function of ϕ and, for each state $s \in S$, an update function $\phi'^\sharp_s \colon p[i] + 1 \to S$ that behaves like the update function of ϕ at s when the input is from $p[i]$ (sending $d \in p[i]$ to $\phi^\sharp_s(d)$) but does not change the state when the input is from 1 (sending the unique element of 1 to s).

This construction is actually universal in a way we'll find important later. Indeed, it's called *copointing*; see Proposition 7.20.

Solution to Exercise 3.33.

We are given $n \in \mathbb{N}$ and an (A_i, B_i)-Moore machine with state set S_i, i.e. a lens $S_i y^{S_i} \to B_i y^{A_i}$, for each $i \in \mathsf{n}$. Taking their parallel product in **Poly** gives us a lens

$$\left(\prod_{i \in \mathsf{n}} S_i \right) y^{\prod_{i \in \mathsf{n}} S_i} \cong \bigotimes_{i \in \mathsf{n}} S_i y^{S_i} \to \bigotimes_{i \in \mathsf{n}} B_i y^{A_i} \cong \left(\prod_{i \in \mathsf{n}} B_i \right) y^{\prod_{i \in \mathsf{n}} A_i},$$

which is a $(\sum_{i \in \mathsf{n}} A_i, \prod_{i \in \mathsf{n}} B_i)$-Moore machine with state set S (after all, the product of monomials is still a monomial).

Solution to Exercise 3.35.

We will show that taking the parallel product of the robot-on-a-grid dynamical system $\phi \colon Sy^S \to p$ from Example 3.28 and a reward-tracking dynamical system $\psi \colon Ty^T \to q$ that we will define yields the dynamical system $\phi' \colon S'y^{S'} \to p'$ from Exercise 3.29.

The reward-tracking dynamical system should have states in $\mathsf{List}(\mathbb{R})$ to record a list of reward values, unchanging output, and inputs in $\mathbb{R}$ to give new reward values. So it is the lens $\mathsf{List}(\mathbb{R})y^{\mathsf{List}(\mathbb{R})} \to y^{\mathbb{R}}$ that has a uniquely defined return function, while its update function sends each state $(r_1, \ldots, r_k) \in \mathsf{List}(\mathbb{R})$ and each input $r \in \mathbb{R}$ to the new state $(r_1, \ldots, r_k, r)$.

Then the dynamical system from Exercise 3.29 is the parallel product of the robot-on-a-grid dynamical system from Example 3.28 with the reward-tracking dynamical system $\mathsf{List}(\mathbb{R})y^{\mathsf{List}(\mathbb{R})} \to y^{\mathbb{R}}$, as can be seen in the solution to Exercise 3.29.

Solution to Exercise 3.36.

1. The robot-on-a-grid dynamical system from Example 3.28 can be written as the parallel product of two robot-on-a-line dynamical systems of the form $\lambda\colon \mathsf{n}y^{\mathsf{n}} \to \sum_{i\in\mathsf{n}} y^{D_i}$, where $\lambda_1 := \mathrm{id}_{\mathsf{n}}$ and $\lambda_i^\sharp$ for each $i \in \mathsf{n}$ sends each direction $d \in D_i$ to the position on the line given by $i + d$. This yields a robot that can move along a single axis, and the parallel product of this robot with itself yields a robot that can move along two different axes at once, which is precisely our robot-on-a-grid dynamical system.
2. To create a dynamical system consisting of a robot moving in a k-dimensional grid of size n along every dimension, we just take the k-fold parallel product of the dynamical system $\lambda\colon \mathsf{n}y^{\mathsf{n}} \to \sum_{i\in\mathsf{n}} y^{D_i}$ we just defined to obtain a dynamical system
$$\lambda^{\otimes k}\colon \mathsf{n}^{\mathsf{k}}y^{\mathsf{n}^{\mathsf{k}}} \to \sum_{(i_1,\ldots,i_k)\in\mathsf{n}^{\mathsf{k}}} y^{\prod_{j\in\mathsf{k}} D_{i_j}}.$$
In fact, we could have used a different n_j for each $j \in \mathsf{k}$ instead of n to obtain a robot moving in an arbitrary k-dimensional grid of size $n_1 \times \cdots \times n_k$ as a k-fold parallel product.

Solution to Exercise 3.38.

We give a lens $f\colon p \to q$ for which composing the file-reader $\phi\colon Sy^S \to p$ from Example 3.26 with f yields the file-searcher $\psi\colon Sy^S \to q$ from Exercise 3.27. The file-searcher returns the same output as the file-reader when the second coordinate is 100, but replaces the second coordinate with a blank _ otherwise. So the on-positions function of f should send each $(m, c) \in p(1)$ to

$$f_1(m,c) = \begin{cases} (m,c) & \text{if } c = 100 \\ (m,\_) & \text{otherwise.} \end{cases}$$

Then the file-searcher acts just like the file-reader does on inputs, so the on-directions functions of f should be id_S at each position whose first coordinate is 'ready' and id_1 at each position whose first coordinate is 'busy.'

Solution to Exercise 3.42.

Given a polynomial p, we wish to show that situations for p are precisely sections of the function $\pi_p\colon \dot{p}(1) \to p(1)$ from Exercise 2.65. That function sends every direction of p to the position at which it is located. So a section $\gamma\colon p(1) \to \dot{p}(1)$ of π_p would send each p-position to a direction of p at that position, which is also exactly what a situation for p does. Alternatively, we can directly apply Exercise 2.18 to deduce that the dependent product $\Gamma(p) \cong \prod_{i\in p(1)} p[i]$ is isomorphic to the set of sections of the projection $\sum_{i\in p(1)} p[i] \to p(1)$ defined by $(i, x) \mapsto i$, which is exactly π_p.

Solution to Exercise 3.43.

1. No, it represents a function $B \to A$! An enclosure assigns each output $b \in B$ to an input $a \in A$.
2. As a wrapper around an interface By^A, an enclosure $\gamma\colon By^A \to y$ corresponds to a function $g\colon B \to A$ that feeds the input $g(b) \in A$ into the system whenever it returns the output $b \in B$.
3. Composing our original Moore machine $Sy^S \to By^A$ with an enclosure γ yields a Moore machine $Sy^S \xrightarrow{\phi} By^A \xrightarrow{\gamma} y$ that returns unchanging output and receives unchanging input. If we identify the Moore machine with its return function $S \to B$ and its update function $S \times A \to S$, and

if we identify the enclosure γ with a function $g\colon B \to A$, then their composite Moore machine $Sy^S \to y$ can be identified with a function $S \to S$, equal to the composite

$$S \xrightarrow{\Delta} S \times S \xrightarrow{\mathrm{id}_S \times \mathrm{return}} S \times B \xrightarrow{\mathrm{id}_S \times g} S \times A \xrightarrow{\mathrm{update}} S,$$

where Δ is the diagonal map $s \mapsto (s, s)$. This composite map $S \to S$ sends every state to the next according to the output the original state returns, the input that the enclosure gives in response to that output, and the update function that sends the original state and the input to the new state.

Solution to Exercise 3.45.

Proposition 3.44 follows directly from Proposition 2.50: we have that $\Gamma(0) = \mathbf{Poly}(0, y) \cong 1$ since 0 is initial in **Poly**, and $\Gamma(p + q) = \mathbf{Poly}(p + q, y) \cong \mathbf{Poly}(p + q, y) = \Gamma(p) \times \Gamma(q)$ since $+$ gives coproducts in **Poly**.

Solution to Exercise 3.49.

We define a new tracker $T'y^{T'} \to \mathbb{N}y^2$ based on the one from Example 3.48 to watch for when the paddle switches sides once, at which point the tracker should increase its location by one, and watch for when the paddle switches sides twice in a row, at which point the tracker should increase its location by two. To do this, we need the tracker to remember not just the current side the paddle is on, but the previous side the paddle was on as well. The tracker should still remember the current location. Thus the states of the tracker are given by $T := 2 \times 2\mathbb{N}$, storing the previous side the paddler was on, the current side the paddler is on, and the current location. The on-positions function of the tracker is the projection $2 \times 2\mathbb{N} \to \mathbb{N}$ that returns the current location; then at each $(d, d', i) \in 2\mathbb{N}$, the on-directions function of the tracker $2 \to 2 \times 2\mathbb{N}$ sends

$$d'' \mapsto \begin{cases} (d', d'', i) & \text{if } d' = d'' \\ (d', d'', i+1) & \text{if } d' \neq d'' \text{ and } d = d' \\ (d', d'', i+2) & \text{if } d' \neq d'' \text{ and } d \neq d' \end{cases}$$

storing both the last side the paddle was on and the new side the paddle is on as well as moving the machine forward one unit if the paddle switches after not switching and two units if the paddle switches after just switching.

Solution to Exercise 3.51.

Here (X, d) is a metric space, A, A' are sets of signals with default signals $a_0 \in A$ and $a'_0 \in A'$, and there are two robots, one with interface $p := AXy^{A'X}$ returning a signal in A and a location in X and another with interface $p' := A'Xy^{AX}$ returning a signal in A' and a location in X.

1. An interaction pattern $p \otimes p' \to y$ consists of a trivial on-positions function $AX \times A'X \to 1$ (indicating that no outputs leave the system) and an on-directions function $AX \times A'X \to A'X \times AX$ indicating what inputs the robots should receive according to the outputs they return. To model the fact that the robots receive each others' locations, but only receive each others' signals rather than the default signals when the distance between their locations is less than 1 according to the distance function d, this on-directions function should send

$$((a, x), (a', x')) \mapsto \begin{cases} ((a', x'), (a, x)) & \text{if } d(x, x') < 1 \\ ((a'_0, x'), (a_0, x)) & \text{otherwise.} \end{cases}$$

2. An interaction pattern $p \otimes p' \to y^{[0,5]}$ that allows the signal to travel $s \in [0, 5]$ times further consists of a still trivial on-positions function and an on-directions function $AX \times A'X \times [0, 5] \to A'X \times AX$ indicating what inputs the robots should receive according to the external input $s \in [0, 5]$ as well as the outputs they return. To model the fact that the robots receive each others' locations, but only receive each others' signals rather than the default signals when the distance between their

locations is less than s according to the distance function d, this on-directions function should send

$$((a,x),(a',x'),s) \mapsto \begin{cases} ((a',x'),(a,x)) & \text{if } d(x,x') < s \\ ((a'_0,x'),(a_0,x)) & \text{otherwise.} \end{cases}$$

3. To provide a dynamical system $\phi\colon SXy^{SX} \to AXy^{A'X}$ under the condition that the on-positions function preserves the second coordinate $x \in X$, we must provide the first projection $SX \to A$ of an on-positions function that turns the robot's private state and current location into the signal it returns, as well as an on-directions function $SX \times A'X \to SX$ that provides a new private state and location for the robot given its old private state, old location, and the signal and location it receives from the other robot.
4. To have the robots listen for each others' signals only when they are sufficiently close, we must move away from monomial interfaces and Moore machines to leverage dependency. There are several ways of doing this; we give just one method below. With $D := \{\text{'close'}, \text{'far'}\}$, let the robots' new interfaces be

$$p := \{\text{'close'}\}AXy^{DA'X} + \{\text{'far'}\}AXy^{DX} \qquad \text{and} \qquad p' := \{\text{'close'}\}A'Xy^{DAX} + \{\text{'far'}\}A'Xy^{DX},$$

so that they may receive input telling them whether they are close or far, but cannot receive signals in A or A' when they are 'far.'

Then by the distributivity of $\otimes$ over $+$, their new interaction pattern $p \otimes p' \to y^{[0,5]}$ can be specified by four lenses, all trivial on positions: the lens

$$\{\text{'close'}\}AXy^{DA'X} \otimes \{\text{'close'}\}A'Xy^{DAX} \to y^{[0,5]},$$

given by the on-directions function

$$((\text{'close'},a,x),(\text{'close'},a',x'),s) \mapsto \begin{cases} ((\text{'close'},a',x'),(\text{'close'},a,x)) & \text{if } d(x,x') < s \\ ((\text{'far'},a'_0,x'),(\text{'far'},a_0,x)) & \text{otherwise;} \end{cases}$$

the lens

$$\{\text{'far'}\}AXy^{X} \otimes \{\text{'far'}\}A'Xy^{X} \to y^{[0,5]},$$

given by the on-directions function

$$((\text{'far'},a,x),(\text{'far'},a',x'),s) \mapsto \begin{cases} ((\text{'close'},x'),(\text{'close'},x)) & \text{if } d(x,x') < s \\ ((\text{'far'},x'),(\text{'far'},x)) & \text{otherwise;} \end{cases}$$

and two other lenses that can be defined arbitrarily, as they should never come up in practice.

Finally, in order for each robot to properly remember whether the other is close or far, we record an element of D in its state that is returned and updated: one robot is a lens

$$\phi\colon DSXy^{DSX} \to \{\text{'close'}\}AXy^{DA'X} + \{\text{'far'}\}AXy^{DX}$$

whose on-positions function preserves not just the third coordinate $x \in X$ but also the first coordinate $d \in D$, while the on-directions function also preserves the first coordinate $d \in D$; and the other robot is constructed similarly.

Solution to Exercise 3.54.

1. A lens $HPy^H \otimes Hy^{HP} \to y$ consists of an on-positions function $HPH \to 1$ and an on-directions function $HPH \times 1 \to HHP$. This amounts to a function $HPH \to HHP$. We can easily define this function to be the isomorphism that sends $(h,p,h') \in HPH$ to (h,h',p).
2. To model the way in which you cycle through three possible actions—reaching down and grabbing the chalk, lifting it with your hand, and dropping it—it is simplest to work with a set of 3 possible states. So we will give your dynamics as a lens $3y^3 \to HPy^H$, where the return function $3 \to HP$ indicates what happens at each state, sending $1 \mapsto$ (down, press), $2 \mapsto$ (up, press), and $3 \mapsto$ (up, no press). Then the update function $3H \to 3$ always goes to the next state, regardless of input: it ignores the H coordinate and sends 1 to 2, 2 to 3, and 3 to 1.

Solution to Exercise 3.58.

We have $p \coloneqq q \coloneqq \mathbb{N}y^{\mathbb{N}}$.

1. If we want to feed the output of q as input to p, the corresponding function $f\colon q(1) \to \Gamma(p)$ should send any output $b \in q(1)$ to the enclosure $\mathbb{N} \to \mathbb{N}$ of p that sends any natural number output of p to b itself. That is, f is the function $b \mapsto (\_ \mapsto b)$.
2. If we want to feed the sum of the outputs of p and q as input to q, the corresponding function $g\colon p(1) \to \Gamma(q)$ should send any output $a \in q(1)$ to the enclosure $\mathbb{N} \to \mathbb{N}$ of q that sends every natural number output b of q to the sum $a + b$. That is, g is the function $a \mapsto (b \mapsto a + b)$.
3. Together, f and g form a function $\mathbb{N}\times\mathbb{N} \to \mathbb{N}\times\mathbb{N}$ mapping $(a, b) \mapsto ((f(b))(a), (g(a))(b)) = (b, a+b)$, which is the enclosure $\gamma\colon p \otimes q \to y$ that (f, g) corresponds to via (3.56).
4. As ϕ and ψ are both the identity, the system $(\phi \otimes \psi) \mathbin{\fatsemi} \gamma$ is really just the system $\gamma\colon p \otimes q \to y$. When it is at state (a, b), its next state will be $(b, a + b)$. So if its initial state is $(0, 1)$, its following states will be $(1, 1), (1, 2), (2, 3), (3, 5), (5, 8), (8, 13), \ldots$, forming the familiar Fibonacci sequence.

Solution to Exercise 3.59.

We wish to write the enclosure $\gamma\colon \texttt{You} \otimes \texttt{Chalk} \to y$ from Example 3.52 as a pair of functions $\texttt{You}(1) \to \Gamma(\texttt{Chalk})$ and $\texttt{Chalk}(1) \to \Gamma(\texttt{You})$ via (3.56).

1. Fix an output $(s_{\texttt{Chalk}}, h_{\text{chalk}}) \in \texttt{Chalk}(1) = \{\text{'out'}, \text{'in'}\}H$ of the chalk. If $s_{\texttt{Chalk}} = \text{'out'}$, then the corresponding enclosure $\texttt{You} \cong HPy^H \to y$ given by f via (3.56) can be thought of as the function $HP \to H$ sending

   $$(h_{\texttt{You}}, p_{\texttt{You}}) \mapsto h_{\texttt{Chalk}},$$

   according to the behavior of $\alpha\colon HPy^H \otimes Hy^P \to y$ when we fix h_{chalk} to be position in the rightmost H and focus on the result in the exponent H on the left. Meanwhile, if $s_{\texttt{Chalk}} = \text{'in'}$, then the corresponding enclosure $\texttt{You} \cong HPy^H \to y$ can also be thought of as the function $HP \to H$ sending

   $$(h_{\texttt{You}}, p_{\texttt{You}}) \mapsto h_{\texttt{Chalk}},$$

   according to the behavior of $\beta\colon HPy^H \otimes Hy^{HP} \to y$ when we fix h_{chalk} to be position in the rightmost H and focus on the result in the exponent H on the left. So overall, the map $\texttt{Chalk}(1) \to \Gamma(\texttt{You})$ sends

   $$(\_, h_{\text{chalk}}) \mapsto ((\_, \_) \mapsto h_{\texttt{Chalk}}).$$

2. Fix an output $(h_{\texttt{You}}, p_{\texttt{You}}) \in \texttt{You}(1) = HP$ of the chalk. Then the corresponding enclosure $\texttt{Chalk} \cong \{\text{'out'}\}Hy^P + \{\text{'in'}\}Hy^{HP} \to y$ can be thought of as a pair of functions: one $\{\text{'out'}\}H \to P$ sending

   $$(\text{'out'}, h_{\texttt{Chalk}}) \mapsto \begin{cases} \text{'no press'} & \text{if } h_{\texttt{You}} \neq h_{\texttt{Chalk}} \\ p_{\texttt{You}} & \text{if } h_{\texttt{You}} = h_{\texttt{Chalk}}. \end{cases}$$

   according to the behavior of $\alpha\colon HPy^H \otimes Hy^P \to y$ when we fix $(h_{\texttt{You}}, p_{\texttt{You}})$ to be position in HP on the left and focus on the result in the exponent P on the right; and another $\{\text{'in'}\}H \to HP$ sending

   $$(\text{'in'}, h_{\texttt{Chalk}}) \mapsto (h_{\texttt{You}}, p_{\texttt{You}})$$

   according to the behavior of $\beta\colon HPy^H \otimes Hy^{HP} \to y$ when we fix $(h_{\texttt{You}}, p_{\texttt{You}})$ to be position in HP on the left and focus on the result in the exponent HP on the right. So overall, the map $\texttt{You}(1) \to \Gamma(\texttt{Chalk})$ sends

   $$(h_{\texttt{You}}, p_{\texttt{You}}) \mapsto \left(\begin{array}{c} (\text{'out'}, h_{\texttt{Chalk}}) \mapsto \begin{cases} \text{'no press'} & \text{if } h_{\texttt{You}} \neq h_{\texttt{Chalk}} \\ p_{\texttt{You}} & \text{if } h_{\texttt{You}} = h_{\texttt{Chalk}} \end{cases} \\ (\text{'in'}, h_{\texttt{Chalk}}) \mapsto (h_{\texttt{You}}, p_{\texttt{You}}) \end{array} \right).$$

Solution to Exercise 3.60.

1. We generalize (3.56) for n polynomials as follows. Given polynomials $p_1, \ldots, p_n \in \mathbf{Poly}$, we claim there is a bijection

$$\Gamma\left(\bigotimes_{i=1}^{n} p_i\right) \cong \prod_{i=1}^{n} \mathbf{Set}\left(\prod_{\substack{1\leq j\leq n,\\ j\neq i}} p_j(1), \Gamma(p_i)\right).$$

The $n = 1$ case is clear, and the $n = 2$ case is given by (3.56). Then by induction on n, we have

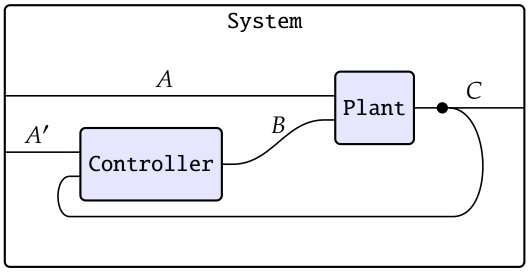
$$\begin{align*}
\Gamma\left(\bigotimes_{i=1}^{n} p_i\right) &\cong \mathbf{Set}\left(p_n(1), \Gamma\left(\bigotimes_{i=1}^{n-1} p_i\right)\right) \times \mathbf{Set}\left(\prod_{i=1}^{n-1} p_i(1), \Gamma(p_n)\right) \tag{3.56}\\
&\cong \mathbf{Set}\left(p_n(1), \prod_{i=1}^{n-1} \mathbf{Set}\left(\prod_{\substack{1\leq j\leq n-1,\\ j\neq i}} p_j(1), \Gamma(p_i)\right)\right) \times \mathbf{Set}\left(\prod_{i=1}^{n-1} p_i(1), \Gamma(p_n)\right) \tag{Inductive hypothesis}\\
&\cong \prod_{i=1}^{n-1} \mathbf{Set}\left(\prod_{\substack{1\leq j\leq n,\\ j\neq i}} p_j(1), \Gamma(p_i)\right) \times \mathbf{Set}\left(\prod_{i=1}^{n-1} p_i(1), \Gamma(p_n)\right), \tag{Universal properties of products and internal homs}
\end{align*}$$

and the result follows.

2. The general idea is that specifying an enclosure for interfaces $p_1, \ldots, p_n$ together is equivalent to specifying an enclosure for p_i for every output all the other interfaces might return, for each $i \in \mathsf{n}$.

Solution to Exercise 3.64.

1. Here is the wiring diagram (3.62) modified so that the controller also receives information of type A' from the outside world.



2. The monomials represented by the boxes in this diagram are the same, except that `Controller` and `System` each have an extra A' exponent:

$$\texttt{Plant} \coloneqq Cy^{AB} \qquad \texttt{Controller} \coloneqq By^{A'C} \qquad \texttt{System} \coloneqq Cy^{AA'}.$$

3. The interaction pattern represented by this wiring diagram is the lens

$$w' \colon \texttt{Plant} \otimes \texttt{Controller} \to \texttt{System}$$

consisting of an on-positions function $CB \to C$ given by $(c, b) \mapsto c$ and an on-directions function $CBAA' \to ABA'C$ given by $(c, b, a, a') \mapsto (a, b, a', c)$.

Solution to Exercise 3.65.

1. According to the wiring diagram, we have that `Alice` $\coloneqq Dy^{HA}$, that `Bob` $\coloneqq EFy^{B}$, and that `Carl` $\coloneqq HGy^{DE}$.

2. According to the wiring diagram, we have that $\texttt{Team} := Gy^{AB}$.
3. The wiring diagram constitutes a wrapper

   $$f\colon \texttt{Alice}\otimes\texttt{Bob}\otimes\texttt{Carl}\to\texttt{Team}.$$

   Its domain is $\texttt{Alice}\otimes\texttt{Bob}\otimes\texttt{Carl}\cong DEFHGy^{HABDE}$, while its codomain is $\texttt{Team} = Gy^{AB}$.
4. On positions, the lens f is a function $DEFHG\to G$ that sends $(d,e,f,h,g)\mapsto g$. On directions, f is a function $DEFHGAB\to HABDE$ that sends $(d,e,f,h,g,a,b)\mapsto(h,a,b,d,e)$.
5. Given dynamical systems $\alpha\colon Ay^{A}\to\texttt{Alice}$, $\beta\colon By^{B}\to\texttt{Bob}$, and $\gamma\colon Cy^{C}\to\texttt{Carl}$, the dynamical system induced by the wiring diagram is given by the composite lens

   $$ABCy^{ABC}\xrightarrow{\alpha\otimes\beta\otimes\gamma}\texttt{Alice}\otimes\texttt{Bob}\otimes\texttt{Carl}\xrightarrow{f}\texttt{Team}.$$

Solution to Exercise 3.66.

1. Using Exercise 3.9, we can turn divmod into the dynamical system divmod: $\mathbb{N}\times\mathbb{N}y^{\mathbb{N}\times\mathbb{N}}\to\mathbb{N}\times\mathbb{N}y^{\mathbb{N}\times\mathbb{N}_{\geq 1}}$ whose return function is the identity on $\mathbb{N}\times\mathbb{N}$ and whose update function $\mathbb{N}\times\mathbb{N}\times\mathbb{N}\times\mathbb{N}_{\geq 1}\to\mathbb{N}\times\mathbb{N}$ sends $(\_,\_,a,b)\mapsto(a \text{ div } b, a \text{ mod } b)$.
2. From left to right, the inner boxes represent monomial interfaces $\mathbb{N}_{\geq 1}y, \mathbb{N}\times\mathbb{N}y^{\mathbb{N}\times\mathbb{N}_{\geq 1}}, \mathbb{N}y$, and $\mathbb{N}y^{\mathbb{N}\times\mathbb{N}}$. The box labeled 7 is given dynamics $7\colon y\to\mathbb{N}_{\geq 1}y$ so that it always returns the output 7; similarly, the box labeled 10 is given dynamics $10\colon y\to\mathbb{N}y$ so that it always returns the output 10. Meanwhile, the box labeled is given dynamics divmod: $\mathbb{N}\times\mathbb{N}y^{\mathbb{N}\times\mathbb{N}}\to\mathbb{N}\times\mathbb{N}y^{\mathbb{N}\times\mathbb{N}_{\geq 1}}$ from the previous part; while we can again apply Exercise 3.9 to the standard multiplication function $*\colon\mathbb{N}\times\mathbb{N}\to\mathbb{N}$ to give the box labeled $*$ dynamics as well, yielding a dynamical system $*\colon\mathbb{N}y^{\mathbb{N}}\to\mathbb{N}y^{\mathbb{N}\times\mathbb{N}}$ whose return function is the identity on $\mathbb{N}$ and whose update function $\mathbb{N}\times\mathbb{N}\times\mathbb{N}\to\mathbb{N}$ sends $(\_,m,n)\mapsto m*n$.

   Then the outer box is the monomial interface $\mathbb{N}\times\mathbb{N}y^{\mathbb{N}}$, and the wiring diagram is the interaction pattern

   $$w\colon\mathbb{N}_{\geq 1}y\otimes\left(\mathbb{N}\times\mathbb{N}y^{\mathbb{N}\times\mathbb{N}_{\geq 1}}\right)\otimes\mathbb{N}y\otimes\mathbb{N}y^{\mathbb{N}\times\mathbb{N}}\to\mathbb{N}\times\mathbb{N}y^{\mathbb{N}}$$

   with on-positions function $(s,q,r,t,p)\mapsto(q,p)$ and on-directions function $(s,q,r,t,p,a)\mapsto(a,s,r,t)$. So the dynamical system induced by the wiring diagram is the composite lens ϕ given by

   $$y\otimes\left(\mathbb{N}\times\mathbb{N}y^{\mathbb{N}\times\mathbb{N}}\right)\otimes y\otimes\mathbb{N}y^{\mathbb{N}}\xrightarrow{7\otimes\text{divmod}\otimes 10\otimes *}\mathbb{N}_{\geq 1}y\otimes\left(\mathbb{N}\times\mathbb{N}y^{\mathbb{N}\times\mathbb{N}_{\geq 1}}\right)\otimes\mathbb{N}y\otimes\mathbb{N}y^{\mathbb{N}\times\mathbb{N}}\xrightarrow{w}\mathbb{N}\times\mathbb{N}y^{\mathbb{N}},$$

   whose return function is given by the composite map $(q,r,p)\mapsto(7,q,r,10,p)\mapsto(q,p)$ and whose update function at state (q,r,p) is given by the composite map $a\mapsto(a,7,r,10)\mapsto(a \text{ div } 7, a \text{ mod } 7, r*10)$.

   In other words, the dynamical system ϕ behaves as follows: its state consists of a quotient q, a remainder r, and a product p, of which it returns the quotient and the product. Then it is fed a dividend a and evaluates a div 7 to obtain the new quotient and a mod 7 to obtain the new remainder. Meanwhile, the new product is given by the previous remainder multiplied by 10.
3. This second wiring diagram specifies an interaction pattern

   $$w'\colon\mathbb{N}\times\mathbb{N}y^{\mathbb{N}}\to\mathbb{N}y$$

   with on-positions function $(q,p)\mapsto q$ and on-directions function $(q,p)\mapsto p$. So the dynamical system induced by nesting the first wiring diagram within the inner box in this second wiring diagram is a composite lens

   $$\left(\mathbb{N}\times\mathbb{N}y^{\mathbb{N}\times\mathbb{N}}\right)\otimes\mathbb{N}y^{\mathbb{N}}\xrightarrow{\phi}\mathbb{N}\times\mathbb{N}y^{\mathbb{N}}\xrightarrow{w'}\mathbb{N}y$$

   whose return function is given by the composite map $(q,r,p)\mapsto(q,p)\mapsto q$ and whose update function at state (q,r,p) is specifies the new state $(p \text{ div } 7, p \text{ mod } 7, r*10)$.

In other words, the dynamical system ϕ behaves as follows: its state consists of a quotient q, a remainder r, and a product p, of which it returns just the quotient. Then it advances to a new state by evaluating p div 7 to obtain the new quotient and p mod 7 to obtain the new remainder. Meanwhile, the new product is given by the previous remainder multiplied by 10.

If the initial state is $(q, r, p) = (0, 0, 10)$, then all the states will be as follows, with the values of q in the left column giving us the outputs:

q (p div 7)	r (p mod 7)	p ($10r$)
0	0	10
1	3	0
0	0	30
4	2	0
0	0	20
2	6	0
0	0	60
8	4	0
0	0	40
5	5	0
0	0	50
7	1	0
0	0	10
...	...	...

Solution to Exercise 3.68.

We seek to model Conway's Game of Life as a closed dynamical system using Example 3.67.

1. Following the suggestion from the end of Example 3.67, we can use a graph with $V := \mathbb{Z} \times \mathbb{Z}$ and $E := (\{-1, 0, 1\} \times \{-1, 0, 1\} - \{(0, 0)\}) \times V$ with $s(i, j, m, n) = (m, n)$ and $t(i, j, m, n) = (m + i, n + j)$ to model cellular automata like Conway's Game of Life on a 2-dimensional integer lattice in which each point listens only to its eight immediate neighbors.
2. Each vertex needs only return whether it is live or dead, so we assign $\tau(v) := \{\text{live}, \text{dead}\}$ for every $v \in V$.
3. For each $v \in V$, the monomial represented by v from Example 3.67 can be written as
$$p_v \cong \{\text{live}, \text{dead}\} y^{\mathbf{Set}(\{-1,0,1\}\times\{-1,0,1\}-\{(0,0)\},\ \{\text{live},\text{dead}\})}.$$
Every vertex returns as output whether it is live or dead and receives as input whether each of its eight neighbors is alive or dead.
4. Each vertex $v \in V$ only needs to record whether it is live or dead, so $S_v := \{\text{live}, \text{dead}\}$.
5. The appropriate dynamical system map $S_v y_v^S \to p_v$ for each vertex $v \in V$ should have the identity function on $\{\text{live}, \text{dead}\}$ as its return function, while its update function should be a map
$$\{\text{live}, \text{dead}\}\mathbf{Set}(\{-1, 0, 1\} \times \{-1, 0, 1\} - \{(0, 0)\},\ \{\text{live}, \text{dead}\}) \to \{\text{live}, \text{dead}\}$$
that executes the rules from Conway's Game of Life, using the data of whether the vertex itself is live or dead as well as whether each of its eight neighbors is live or dead to determine whether it should be live or dead in the next time step.

Solution to Exercise 3.71.

1.
2.

Solution to Exercise 3.78.

We compute $[q, r]$ for various values of $q, r \in \mathbf{Poly}$ using (3.77).

1. If $q := 0$, then $q(1) \cong 0$, so $[q, r]$ is an empty product. Hence $[q, r] \cong 1$.

2. If $q := 1$, then $q(1) \cong 1$ and $q[1] \cong 0$, so $[q, r] \cong r \circ (0y) \cong r(0)$.
3. If $q := y$, then $q(1) \cong 1$ and $q[1] \cong 1$, so $[q, r] \cong r \circ (1y) \cong r$.
4. If $q := A$ for $A \in \mathbf{Set}$, then $q(1) \cong A$ and $q[j] \cong 0$ for every $j \in A$, so $[q, r] \cong \prod_{j\in A}(r \circ (0y)) \cong r(0)^A$.
5. If $q := Ay$ for $A \in \mathbf{Set}$, then $q(1) \cong A$ and $q[j] \cong 1$ for every $j \in A$, so $[q, r] \cong \prod_{j\in A}(r \circ (1y)) \cong r^A$.
6. If $q := y^2 + 2y$ and $r := 2y^3 + 3$, then

$$\begin{aligned}
[q, r] &\cong (r \circ (2y))(r \circ (1y))^2 \\
&\cong \left(2(2y)^3 + 3\right)\left(2y^3 + 3\right)^2 \\
&\cong 64y^9 + 204y^6 + 180y^3 + 27
\end{aligned}$$

Solution to Exercise 3.79.

We wish to show that for all $p_1, p_2, q \in \mathbf{Poly}$, we have $[p_1 + p_2, q] \cong [p_1, q] \times [p_2, q]$. By (3.77),

$$[p_1 + p_2, q] \cong \left(\prod_{i\in p_1(1)} q \circ (p_1[i]y)\right)\left(\prod_{i\in p_2(1)} q \circ (p_2[i]y)\right) \cong [p_1, q] \times [p_2, q].$$

Solution to Exercise 3.80.

We may compute

$$\begin{aligned}
[q, r] &\cong \prod_{j\in q(1)} r \circ (q[j]y) && (3.77) \\
&\cong \prod_{j\in q(1)} \sum_{k\in r(1)} (q[j]y)^{r[k]} && \text{(Replacing each } y \text{ in } r \text{ by } q[j]y\text{)} \\
&\cong \sum_{f_1\colon q(1)\to r(1)} \prod_{j\in q(1)} (q[j]y)^{r[f_1(j)]} && (2.26) \\
&\cong \sum_{f_1\colon q(1)\to r(1)} \left(\prod_{j\in q(1)} q[j]^{r[f_1(j)]}\right)\left(\prod_{j\in q(1)} y^{r[f_1(j)]}\right) \\
&\cong \sum_{f_1\colon q(1)\to r(1)} \sum_{f^\sharp\in\prod_{j\in q(1)} q[j]^{r[f_1(j)]}} y^{\sum_{j\in q(1)} r[f_1(j)]} \\
&\cong \sum_{f\colon q\to r} y^{\sum_{j\in q(1)} r[f_1(j)]}. && (2.54)
\end{aligned}$$

Solution to Exercise 3.82.

We verify (2.67) as follows:

$$\begin{aligned}
[p, y] \otimes p &\cong \left(\sum_{f\colon p\to y} y^{\sum_{i\in p(1)} y[f_1(i)]}\right) \otimes p && (3.81) \\
&\cong \sum_{f\in\Gamma(p)} y^{p(1)} \otimes \sum_{i\in p(1)} y^{p[i]} \\
&\cong \sum_{f\in\Gamma(p)} \sum_{i\in p(1)} y^{p(1)\times p[i]} && (2.89) \\
&\cong \sum_{f\in\prod_{i\in p(1)} p[i]} \sum_{i\in p(1)} y^{p(1)\times p[i]}. && (3.40)
\end{aligned}$$

Solution to Exercise 3.85.

We have that

$$[y^A, y] \cong \prod_{j \in y^A(1)} y \circ (y^A[j]y) \cong \prod_{j \in 1} Ay \cong Ay,$$

that

$$[Ay, y] \cong \prod_{j \in Ay(1)} y \circ ((Ay)[j]y) \cong \prod_{j \in A} y \cong y^A,$$

and that

$$\begin{aligned} [p, y] &\cong \sum_{f \colon p \to y} y^{\sum_{i \in p(1)} y[f_1(i)]} && \text{(Exercise 3.80)} \\ &\cong \sum_{f \in \Gamma(p)} y^{\sum_{i \in p(1)} 1} \\ &\cong \Gamma(p) y^{p(1)}. \end{aligned}$$

Solution to Exercise 3.86.

Given $p \in \mathbf{Poly}$ and an isomorphism $[[p, y], y] \cong p$, we wish to show that p is either linear or representable. Applying (3.84) twice, we have that

$$p \cong [[p, y], y] \cong \Gamma\left(\Gamma(p) y^{p(1)}\right) y^{\Gamma(p)}.$$

We can compute the coefficient of p via (3.40) to obtain

$$\Gamma\left(\Gamma(p) y^{p(1)}\right) \cong \prod_{\gamma \in \Gamma(p)} p(1) \cong p(1)^{\Gamma(p)}.$$

Hence

$$p \cong p(1)^{\Gamma(p)} y^{\Gamma(p)}. \tag{3.96}$$

In particular, p is a monomial, so we can write $p \coloneqq By^A$ for some $A, B \in \mathbf{Set}$. Then $p(1) \cong B$ and (3.40) tells us that $\Gamma(p) \cong A^B$. It follows from (3.96) that $A \cong A^B$ and that $B \cong B^A$.

We conclude with some elementary set theory. If either one of A or B were 1, then p would be either linear or representable, and we would be done. Meanwhile, if either one of A or B were 0, then the other would be 1, and we would again be done. Otherwise, $|A|, |B| \geq 2$. But by Cantor's theorem,

$$|B| < \left|2^B\right| \leq \left|A^B\right| = |A| \qquad \text{and} \qquad |A| < \left|2^A\right| \leq \left|B^A\right| = |B|,$$

a contradiction.

Solution to Exercise 3.89.

The isomorphism $\mathbf{Poly}(p, q) \cong [p, q](1)$ follows directly from Exercise 3.80 when both sides are applied to 1. Alternatively, we can apply (3.88). Since $p \cong y \otimes p$, we have that

$$\begin{aligned} \mathbf{Poly}(p, q) &\cong \mathbf{Poly}(y \otimes p, q) \\ &\cong \mathbf{Poly}(y, [p, q]) && (3.88) \\ &\cong [p, q](1). && \text{(Yoneda lemma)} \end{aligned}$$

Solution to Exercise 3.91.

To obtain the evaluation map $\mathrm{eval} \colon [p, q] \otimes p \longrightarrow q$, we consider the following special case of the isomorphism (3.88):

$$\mathbf{Poly}([p, q] \otimes p, q) \cong \mathbf{Poly}([p, q], [p, q]).$$

Then the evaluation map is the map corresponding to the identity morphism on $[p, q]$ under the above isomorphism. To recover this map, we can start from the identity morphism on $[p, q]$ and work our way along a chain of natural isomorphisms from $\mathbf{Poly}([p,q],[p,q])$ until we get to $\mathbf{Poly}([p,q]\otimes p, q)$. To start, Exercise 3.80 implies that

$$\begin{aligned}\mathbf{Poly}([p,q],[p,q]) &\cong \mathbf{Poly}\left(\sum_{f\colon p\to q}\prod_{i'\in p(1)} y^{q[f_1(i')]}, \prod_{i\in p(1)}\sum_{j\in q(1)}(p[i]y)^{q[j]}\right)\\ &\cong \prod_{f\colon p\to q}\prod_{i\in p(1)}\mathbf{Poly}\left(\prod_{i'\in p(1)} y^{q[f_1(i')]}, \sum_{j\in q(1)}(p[i]y)^{q[j]}\right),\end{aligned}$$

where the second isomorphism follows from the universal properties of products and coproducts. In particular, under this isomorphism, the identity morphism on $[p,q]$ corresponds to a collection of morphisms, namely for each $f\colon p\to q$ and each $i\in p(1)$ the composite

$$\prod_{i'\in p(1)} y^{q[f_1(i')]} \to y^{q[f_1(i)]} \to \sum_{g\colon q[f_1(i)]\to p[i]} y^{q[f_1(i)]} \cong (p[i]y)^{q[f_1(i)]} \to \sum_{j\in q(1)}(p[i]y)^{q[j]}$$

of the canonical projection with index $i'=i$, the canonical inclusion with index $g=f_i^\sharp$, and the canonical inclusion with index $j=f_1(i)$. On positions, this map picks out the position of $\sum_{j\in q(1)}(p[i]y)^{q[j]}$ corresponding to $j=f_1(i)\in q(1)$ and $f_i^\sharp\colon q[f_1(i)]\to p[i]$; on directions, the map is the canonical inclusion $q[f_1(i)]\to\sum_{i'\in p(1)} q[f_1(i')]$ with index $i'=i$.
We can reinterpret each of these maps as a map

$$y^{p[i]\times\sum_{i'\in p(1)} q[f_1(i')]} \to \sum_{j\in q(1)} y^{q[j]} \cong q$$

that, on positions, picks out the position $f_1(i)\in q(1)$ of q and, on directions, is the map $q[f_1(i)]\to p[i]\times\sum_{i'\in p(1)} q[f_1(i')]$ induced by the universal property of products applied to the map $f_i^\sharp\colon q[f_1(i)]\to p[i]$ and the inclusion $q[f_1(i)]\to\sum_{i'\in p(1)} q[f_1(i')]$. Then by the universal property of coproducts, this collection of maps induces a single map $\mathrm{eval}\colon [p,q]\otimes p\to q$ that sends each position $f\colon p\to q$ of $[p,q]$ and position $i\in p(1)$ of p to the position $f_1(i)$ of q, with the same behavior on directions as the corresponding map described previously.

Solution to Exercise 3.92.

1. Given a set S, we wish to obtain the map $Sy^S\to y$ whose on-directions map is the identity by using eval and Example 3.83. The example shows that

$$[Sy,y]\otimes(Sy)\cong y^S\otimes(Sy)\cong Sy^S,$$

so by setting $p\coloneqq Sy$ and $q\coloneqq S$ in (3.90), we obtain an evaluation map $\mathrm{eval}\colon Sy^S\to y$. By the solution to Exercise 3.91, given a position $s\in S$ of Sy^S, the evaluation map on directions is the map $1\to S$ that picks out s. In other words, it is indeed the identity on directions.

2. We wish to write the four maps in (3.93) from Example 3.72 as the parallel product of identity maps and eval maps. By the solution to Exercise 3.91, the eval map $[F,y^F]\otimes F\to y^F$ is a map from

$$[F,y^F]\otimes F\cong F\left(\sum_{f\colon F\to y^F}\prod_{i\in F} y^F\right)\cong Fy^{FF}$$

to y^F that is unique on positions and has the on-directions function $FF\to FF$ given by the identity. Then we can verify that κ_{11} is equivalent to the parallel product of this eval map with itself. We can define κ_{12} and κ_{21} to be the parallel product of this eval map with the identity on y^F, while κ_{22} is the parallel product of the identity on y^F with itself.

Chapter 4

More categorical properties of polynomials

The category **Poly** has very useful formal properties, including completion under colimits and limits, various adjunctions with **Set**, factorization systems, and so on. Most of the following material is not necessary for the development of our main story, but we collect it here for reference. The reader can skip directly to Part II if so inclined, and we will direct them back here when needed. Better yet might be to just gently leaf through Chapter 4, to see how well-behaved and versatile the category **Poly** really is.

4.1 Special polynomials and adjunctions

There are a few special classes of polynomials that are worth discussing:

a) constant polynomials $0, 1, 2, A$;

b) linear polynomials $0, y, 2y, Ay$;

c) pure-power (or representable) polynomials $1, y, y^2, y^A$; and

d) monomials $0, A, y, 2y^3, Ay^B$.

The first two classes, constant and linear polynomials, are interesting because they both put a copy of **Set** inside **Poly**, as we'll see in Propositions 4.2 and 4.3. The third puts a copy of $\mathbf{Set}^{\mathrm{op}}$ inside **Poly**: it is the Yoneda embedding that we saw way back in Exercise 2.12. Finally, the fourth puts a copy of bimorphic lenses inside **Poly**, as we saw in **??**.

Exercise 4.1 (Solution here). Which of the four classes above are closed under

1. the cocartesian monoidal structure $(0, +)$ (i.e. addition)?
2. the cartesian monoidal structure $(1, \times)$ (i.e. multiplication)?
3. the parallel monoidal structure $(y, \otimes)$ (i.e. taking the parallel product)?
4. composition of polynomials $p \circ q$? ◊

Proposition 4.2. There is a fully faithful functor **Set** → **Poly** sending $A \mapsto Ay^0 = A$.

Proof. By (2.55), a map $f\colon Ay^0 \to By^0$ consists of a function $f\colon A \to B$ and, for each $a \in A$, a function $0 \to 0$. There is only one function $0 \to 0$, so f can be identified with just a map of sets $A \to B$. □

Proposition 4.3. There is a fully faithful functor **Set** → **Poly** sending $A \mapsto Ay$.

Proof. By (2.55), a map $f\colon Ay^1 \to By^1$ consists of a function $f\colon A \to B$ and for each $a \in A$ a function $1 \to 1$. There is only one function $1 \to 1$, so f can be identified with just a map of sets $A \to B$. □

Theorem 4.4. **Poly** has an adjoint quadruple with **Set**:

$$\begin{tikzcd}
\mathbf{Set} \arrow[from=r, shift right=6, "p(0)"'] \arrow[r, phantom, shift left=4, "\Rightarrow"] \arrow[r, shift left=2, "A" description] \arrow[r, phantom, "\Leftarrow"] \arrow[from=r, shift left=2, "p(1)" description] \arrow[r, phantom, shift right=4, "\Rightarrow"] \arrow[r, shift right=6, "Ay"'] & \mathbf{Poly}
\end{tikzcd} \tag{4.5}$$

where the functors have been labeled by where they send $A \in$ **Set** and $p \in$ **Poly**.

Both rightward functors are fully faithful.

Proof. For any set A, there is a functor **Poly** → **Set** given by sending p to $p(A)$; by the Yoneda lemma, it is the functor $\mathbf{Poly}(y^A, -)$. This, together with Propositions 4.2 and 4.3, gives us the four functors and the fact that the two rightward functors are fully faithful. It remains to provide the following three natural isomorphisms:

$$\mathbf{Poly}(A, p) \cong \mathbf{Set}(A, p(0)) \qquad \mathbf{Poly}(p, A) \cong \mathbf{Set}(p(1), A) \qquad \mathbf{Poly}(Ay, p) \cong \mathbf{Set}(A, p(1)).$$

All three come from our formula (2.54) for computing general hom-sets in **Poly**; we leave the details to the reader in Exercise 4.6. □

Exercise 4.6 (Solution here). Here we prove the remainder of Theorem 4.4 using (2.54):

1. Provide a natural isomorphism $\mathbf{Poly}(A, p) \cong \mathbf{Set}(A, p(0))$.
2. Provide a natural isomorphism $\mathbf{Poly}(p, A) \cong \mathbf{Set}(p(1), A)$.
3. Provide a natural isomorphism $\mathbf{Poly}(Ay, p) \cong \mathbf{Set}(A, p(1))$. ◊

Exercise 4.7 (Solution here). Show that for any polynomial p, its set $p(1)$ of positions is in bijection with the set of functions $y \to p$. ◊

In Theorem 4.4 we see that $p \mapsto p(0)$ and $p \mapsto p(1)$ have left adjoints. This is true more generally for any set A in place of 0 and 1, as we show in Corollary 4.10. However, the fact that $p \mapsto p(1)$ is itself the left adjoint of the left adjoint of $p \mapsto p(0)$—and hence that we have the *quadruple* of adjunctions in (4.5)—is special to $A = 0, 1$.

We also have a copower-hom-power two-variable adjunction between **Poly**, **Set**, and **Poly**.

Proposition 4.8. There is a two-variable adjunction between **Poly**, **Set**, and **Poly**:

$$\mathbf{Poly}(Ap, q) \cong \mathbf{Set}(A, \mathbf{Poly}(p, q)) \cong \mathbf{Poly}(p, q^A). \tag{4.9}$$

Proof. Since Ap is the A-fold coproduct of p and q^A is the A-fold product of q, the universal properties of coproducts and products give natural isomorphisms

$$\mathbf{Poly}(Ap, q) \cong \prod_{a \in A} \mathbf{Poly}(p, q) \cong \mathbf{Poly}(p, q^A).$$

The middle set is naturally isomorphic to $\mathbf{Set}(A, \mathbf{Poly}(p, q))$, completing the proof. □

Replacing p with y^B in (4.9), we obtain the following using the Yoneda lemma.

Corollary 4.10. For any set B there is an adjunction

$$\begin{tikzcd} \mathbf{Set} \arrow[r, shift left=1.5ex, "Ay^B"] \arrow[r, phantom, "\Rightarrow"] & \mathbf{Poly} \arrow[l, shift left=1.5ex, "q(B)"] \end{tikzcd}$$

where the functors are labeled by where they send $q \in \mathbf{Poly}$ and $A \in \mathbf{Set}$.

Exercise 4.11 (Solution here). Prove Corollary 4.10 from Proposition 4.8. ◊

Proposition 4.12. The Yoneda embedding $A \mapsto y^A$ has a left adjoint

$$\begin{tikzcd} \mathbf{Set}^{\mathrm{op}} \arrow[r, shift left=1.5ex, "y^{-}"] \arrow[r, phantom, "\Leftarrow"] & \mathbf{Poly} \arrow[l, shift left=1.5ex, "\Gamma"] \end{tikzcd}$$

where $\Gamma(p) := \mathbf{Poly}(p, y) \cong \prod_{i \in p(1)} p[i]$, as in (3.39) and (3.40). That is, there is a natural isomorphism

$$\mathbf{Poly}(p, y^A) \cong \mathbf{Set}(A, \Gamma(p)). \tag{4.13}$$

Proof. By (2.54), we have the natural isomorphism

$$\mathbf{Poly}(p, y^A) \cong \prod_{i \in p(1)} p[i]^A,$$

which in turn is naturally isomorphic to $\mathbf{Set}(A, \Gamma(p))$ by (3.40). □

Corollary 4.14 (Principle monomial)**.** There is an adjunction

$$\begin{tikzcd} \mathbf{Poly} \arrow[r, shift left=1.5, "{(p(1),\Gamma(p))}"] \arrow[r, phantom, "\Rightarrow"] & \mathbf{Set}\times\mathbf{Set}^{\mathrm{op}} \arrow[l, shift left=1.5, "Ay^B"] \end{tikzcd}$$

where the functors are labeled by where they send $p \in \mathbf{Poly}$ and $(A, B) \in \mathbf{Set}\times\mathbf{Set}^{\mathrm{op}}$. That is, there is a natural isomorphism

$$\mathbf{Poly}(p, Ay^B) \cong \mathbf{Set}(p(1), A) \times \mathbf{Set}(B, \Gamma(p)). \tag{4.15}$$

Proof. By the universal property of the product of A and y^B, we have a natural isomorphism

$$\mathbf{Poly}(p, Ay^B) \cong \mathbf{Poly}(p, A) \times \mathbf{Poly}(p, y^B).$$

Then the desired natural isomorphism follows from Exercise 4.6 #2 and (4.13). □

Exercise 4.16 (Solution here). Use (4.15) together with (3.84) and (3.88) to find an alternative proof for Proposition 3.55, i.e. that there is an isomorphism

$$\Gamma(p \otimes q) \cong \mathbf{Set}\big(q(1), \Gamma(p)\big) \times \mathbf{Set}\big(p(1), \Gamma(q)\big).$$

for any $p, q \in \mathbf{Poly}$. ◊

4.2 Epi-mono factorization

Proposition 4.17. Let $f : p \to q$ be a morphism in **Poly**. It is a monomorphism if and only if the on-positions function $f_1 : p(1) \to q(1)$ is a monomorphism in **Set** and, for each $i \in p(1)$, the on-directions function $f_i^\sharp : q[f_1(i)] \to p[i]$ is an epimorphism in **Set**.

Proof. To prove the forward direction, suppose that f is a monomorphism. Since $p \mapsto p(1)$ is a right adjoint (Theorem 4.4), it preserves monomorphisms, so the on-positions function f_1 is also a monomorphism.

We now need to show that for any $i \in p(1)$, the on-directions function $f_i^\sharp : q[f_1(i)] \to p[i]$ is an epimorphism. Suppose we are given a set A and a pair of functions $g^\sharp, h^\sharp : p[i] \rightrightarrows A$ with $f_i^\sharp \mathbin{⨟} g^\sharp = f_i^\sharp \mathbin{⨟} h^\sharp$. Then there exist morphisms $g, h : y^A \rightrightarrows p$ whose on-positions functions both pick out i and whose on-directions functions are $g^\sharp$ and $h^\sharp$, so that $g \mathbin{⨟} f = h \mathbin{⨟} f$. As f is a monomorphism, $g = h$; in particular, their on-directions functions $g^\sharp$ and $h^\sharp$ are equal, as desired.

Conversely, suppose that f_1 is a monomorphism and that, for each $i \in p(1)$, the function $f_i^\sharp$ is an epimorphism. Let r be a polynomial and $g, h : r \rightrightarrows p$ be two morphisms such that $g \mathbin{⨟} f = h \mathbin{⨟} f$. Then $g_1 \mathbin{⨟} f_1 = h_1 \mathbin{⨟} f_1$, which implies $g_1 = h_1$; we'll consider g_1

the default representation. We also have that $f^\sharp_{g_1(k)} \mathbin{⨟} g^\sharp_k = f^\sharp_{g_1(k)} \mathbin{⨟} h^\sharp_k$ for any $k \in r(1)$. But $f^\sharp_{g_1(k)}$ is an epimorphism, so in fact $g^\sharp_k = h^\sharp_k$, as desired. □

Example 4.18. Choose a finite nonempty set k for $1 \leq k \in \mathbb{N}$, e.g. $\mathsf{k} = \mathsf{12}$. There is a monomorphism

$$f \colon \mathsf{k}y^{\mathsf{k}} \to \mathbb{N}y^{\mathbb{N}}$$

such that the trajectory "going around and around the k-clock" comes from the usual counting trajectory Example 3.5 $\mathbb{N}y^{\mathbb{N}} \to y$.

On positions, we have $f_1(i) = i$ for all $i \in \mathsf{k}$. On directions, for any $i \in \mathsf{k}$, we have $f^\sharp_i(n) = n \mod k$ for all $n \in \mathbb{N}$.

Exercise 4.19 (Solution here). In Example 4.18, we gave a map $\mathsf{12}y^{\mathsf{12}} \to \mathbb{N}y^{\mathbb{N}}$. This allows us to turn any dynamical system with $\mathbb{N}$-many states into a dynamical system with 12 states, while keeping the same interface—say, p.

Explain how the behavior of the new system $\mathsf{12}y^{\mathsf{12}} \to p$ would be seen to relate to the behavior of the old system $\mathbb{N}y^{\mathbb{N}} \to p$. ◊

Proposition 4.20. Let $f \colon p \to q$ be a morphism in **Poly**. It is an epimorphism if and only if the function $f_1 \colon p(1) \to q(1)$ is an epimorphism in **Set** and, for each $j \in q(1)$, the induced function

$$f^\flat_j \colon q[j] \to \prod_{\substack{i \in p(1),\\ f_1(i)=j}} p[i]$$

from (2.62) is a monomorphism.

Proof. To prove the forward direction, suppose that f is an epimorphism. Since $p \mapsto p(1)$ is a left adjoint (Theorem 4.4), it preserves epimorphisms, so the on-positions function f_1 is also a epimorphism.

We now need to show that for any $j \in q(1)$, the induced function $f^\flat_j$ is a monomorphism. Suppose we are given a set A and a pair of functions $g', h' \colon A \rightrightarrows q[j]$ with $g' \mathbin{⨟} f^\flat_j = h' \mathbin{⨟} f^\flat_j$. They can be identified with morphisms $g, h \colon q \rightrightarrows y^A + 1$, which send the j-component to the first component, y^A, and send all other component to the second component, 1. It is easy to check that $fg = fh$, hence $g = h$, and hence $g^\sharp = h^\sharp$ as desired.

Then we can construct morphisms $g, h \colon q \rightrightarrows y^A + 1$ whose on-positions functions both send j to the first position, corresponding to y^A, and all other positions to the second position, corresponding to 1. In addition, we let the on-directions functions be $g^\sharp_j \coloneqq g'$ and $h^\sharp_j \coloneqq h'$. Then $f \mathbin{⨟} g = f \mathbin{⨟} h$. As f is an epimorphism, $g = h$; in particular, their on-directions functions are equal, so $g' = h'$, as desired.

Conversely, suppose that f_1 is an epimorphism and that, for each $j \in q(1)$, the function $f_j^\flat$ is a monomorphism. Let r be a polynomial and $g, h \colon q \rightrightarrows r$ be two morphisms such that $f \fatsemi g = f \fatsemi h$. Then $f_1 \fatsemi g_1 = f_1 \fatsemi h_1$, which implies $g_1 = h_1$; we'll consider g_1 the default representation. We also have that $g^\sharp_{f_1(i)} \fatsemi f^\sharp_i = h^\sharp_{f_1(i)} \fatsemi f^\sharp_i$ for any $i \in p(1)$. It follows that, for any $j \in q(1)$, the two composites

$$\begin{tikzcd}
r[g_1(j)] \arrow[r, shift left, "g_j^\sharp"] \arrow[r, shift right, "h_j^\sharp"'] & q[j] \arrow[r, "f_j^\flat"] & \displaystyle\prod_{\substack{i \in p(1),\\ f_1(i)=j}} p[i]
\end{tikzcd}$$

are equal, which implies that $g_j^\sharp = h_j^\sharp$ as desired. □

Exercise 4.21 (Solution here). Show that the only way for a map $p \to y$ to *not* be an epimorphism is when $p = 0$. ◊

Exercise 4.22 (Solution here). Let A and B be sets and AB their product. Find an epimorphism $y^A + y^B \twoheadrightarrow y^{AB}$. ◊

Exercise 4.23 (Solution here). Suppose a lens is both a monomorphism and an epimorphism; it is then an isomorphism? (That is, is **Poly** *balanced*?)

Hint: You may use the following facts.

1. A function that is both a monomorphism and an epimorphism in **Set** is an isomorphism.
2. A lens is an isomorphism if and only if the on-positions function is an isomorphism and every on-directions function is an isomorphism.

◊

We are often interested in whether epimorphisms and monomorphisms form what is called a *factorization system* in a given category, which we define below.

Definition 4.24 (Factorization system)**.** Given a category $\mathcal{C}$ and two classes of morphisms E and M in $\mathcal{C}$, we say that (E, M) is a *factorization system* of $\mathcal{C}$ if:

1. every morphism f in $\mathcal{C}$ factors uniquely (up to unique isomorphism) as a morphism $e \in E$ composed with a morphism $m \in M$, so that $f = e \fatsemi m$;
2. E and M each contain every isomorphism; and
3. E and M are each closed under composition.

If E is the class of epimorphisms and M is the class of monomorphisms (in which case conditions 2 and 3 are automatically satisfied), we say that $\mathcal{C}$ has *epi-mono factorization*.

Example 4.25 (Epi-mono factorization in **Set**). The category **Set** has epi-mono factorization: a function $f\colon X \to Y$ can be uniquely factored into an epimorphism (surjection) e followed by a monomorphism (injection) i, as follows. The epimorphism $e\colon X \to f(X)$ is given by restricting the codomain of f to its image (also known as *corestricting* f), so e sends $x \mapsto f(x)$ for all $x \in X$. The monomorphism $i\colon f(X) \to Y$ is then given by including the image into the codomain, so i sends $y \mapsto y$ for all $y \in f(X) \subseteq Y$.

Proposition 4.26. **Poly** has epi-mono factorization.

Proof. Take an arbitrary lens $\phi\colon p \to q$. It suffices to show that there exists a unique polynomial r equipped with an epimorphism $\epsilon\colon p \to r$ and a monomorphism $\mu\colon r \to q$ such that $\phi = \epsilon \mathbin{⨟} \mu$.

On positions, we must have $\phi_1 = \epsilon_1 \mathbin{⨟} \mu_1$, with μ_1 a monomorphism and ϵ_1 an epimorphism per Propositions 4.17 and 4.20. By Example 4.25, since **Set** has epi-mono factorization, such $r(1), \epsilon_1$, and μ_1 uniquely exist. In particular, we must have that $r(1) \cong \phi_1(p(1))$, that $\epsilon_1\colon p(1) \to \phi_1(p(1))$ is the corestriction of ϕ_1 sending $i \mapsto \phi_1(i)$ for each p-position i, and that $\mu_1\colon \phi_1(p(1)) \to q(1)$ is the inclusion sending $j \mapsto j$ for each r-position j.

Then on directions, for any $i \in p(1)$, we must have that

$$\begin{tikzcd}
q[\phi_1(i)] \arrow[r, "\mu^\sharp_{\phi_1(i)}"] \arrow[dr, "\phi^\sharp_i"'] & r[\phi_1(i)] \arrow[d, "\epsilon^\sharp_i"] \\
& p[i]
\end{tikzcd}$$

commutes—or, equivalently, for every $j \in r(1) \cong \phi_1(p(1))$,

$$\begin{tikzcd}
q[j] \arrow[r, "\mu^\sharp_j"] \arrow[dr, "\phi^\flat_j"'] & r[j] \arrow[d, "\epsilon^\flat_j"] \\
& \displaystyle\prod_{\substack{i\in p(1),\\ \phi_1(i)=j}} p[i]
\end{tikzcd}$$

commutes (here $\phi^\flat_j$ and $\epsilon^\flat_j$ are the induced functions from (2.62)), with $\mu^\sharp_j$ an epimorphism and $\epsilon^\flat_j$ a monomorphism per Propositions 4.17 and 4.20. So again since **Set** has epi-mono factorization, such $r[j], \mu^\sharp_j$, and $\epsilon^\flat_j$ uniquely exist. Hence such $p \xrightarrow{\epsilon} r \xrightarrow{\mu} q$ uniquely exists overall. □

4.3 Cartesian closure

We have already seen the closure operation $[-,-]$ for one monoidal structure on **Poly**, namely $(y,\otimes)$. But this is not the only closed monoidal structure on **Poly**: in fact, we will show that **Poly** is cartesian closed as well.

For any two polynomials q, r, define $r^q \in \mathbf{Poly}$ by the formula

$$r^q := \prod_{j\in q(1)} r \circ (y + q[j]) \tag{4.27}$$

where $\circ$ denotes composition.

Before proving that this really is an exponential in **Poly**, which we do in Theorem 4.30, we first get some practice with it.

Example 4.28. Let A be a set. We've been writing the polynomial Ay^0 simply as A, so it better be true that the there is an isomorphism

$$y^A \cong y^{Ay^0}$$

in order for the notation to be consistent. Luckily, this is true. By (4.27), we have

$$y^{Ay^0} = \prod_{a\in A} y \circ (y + 0) \cong y^A$$

Exercise 4.29 (Solution here). Compute the following exponentials in **Poly** using (4.27):

1. p^0 for an arbitrary $p \in \mathbf{Poly}$.
2. p^1 for an arbitrary $p \in \mathbf{Poly}$.
3. 1^p for an arbitrary $p \in \mathbf{Poly}$.
4. A^p for an arbitrary $p \in \mathbf{Poly}$ and $A \in \mathbf{Set}$.
5. y^y.
6. y^{4y}.
7. $(y^A)^{y^B}$ for arbitrary sets $A, B \in \mathbf{Set}$. ◊

Theorem 4.30. The category **Poly** is cartesian closed. That is, we have a natural isomorphism

$$\mathbf{Poly}(p, r^q) \cong \mathbf{Poly}(p \times q, r),$$

where r^q is the polynomial defined in (4.27).

Proof. We have the following chain of natural isomorphisms:

$$\mathbf{Poly}(p, r^q) \cong \mathbf{Poly}\Big(p, \prod_{j\in q(1)} r \circ (y + q[j])\Big) \tag{4.27}$$

$$
\begin{aligned}
&\cong \prod_{i\in p(1)} \prod_{j\in q(1)} \mathbf{Poly}\big(y^{p[i]}, r \circ (y + q[j])\big) \\
&\hspace{12em} \text{(Universal property of (co)products)} \\
&\cong \prod_{i\in p(1)} \prod_{j\in q(1)} r \circ (p[i] + q[j]) \qquad \text{(Yoneda lemma)} \\
&\cong \prod_{i\in p(1)} \prod_{j\in q(1)} \sum_{k\in r(1)} (p[i] + q[j])^{r[k]} \\
&\cong \prod_{(i,j)\in(p\times q)(1)} \sum_{k\in r(1)} (p\times q)[(i,j)]^{r[k]} \qquad (2.84) \\
&\cong \mathbf{Poly}(p\times q, r). \qquad (2.54)
\end{aligned}
$$

□

Exercise 4.31 (Solution here). Use Theorem 4.30 to show that for any polynomials p, q, there is a canonical evaluation map

$$\text{eval}\colon p^q \times q \to p.$$

◊

4.4 Limits and colimits

We have already seen that **Poly** has all coproducts (Proposition 2.50) and products (Proposition 2.79). We will now see that **Poly** has all small limits and colimits.

Theorem 4.32. The category **Poly** has all small limits.

Proof. A category has all small limits if and only if it has products and equalizers, so by Proposition 2.79, it suffices to show that **Poly** has equalizers.

We claim that equalizers in **Poly** are simply equalizers on positions and coequalizers on directions. More precisely, let $f, g\colon p \rightrightarrows q$ be two lenses. We construct the equalizer p' of f and g as follows.[1] We define its position-set $p'(1)$ to be the equalizer of $f_1, g_1\colon p(1) \rightrightarrows q(1)$ in **Set**; that is,

$$p'(1) \coloneqq \{i \in p(1) \mid f_1(i) = g_1(i)\}.$$

Then for each $i \in p'(1)$, we can define the direction-set $p'[i]$ to be the coequalizer of $f_i^\sharp, g_i^\sharp\colon q[f_1(i)] \rightrightarrows p[i]$. In this way, we obtain a polynomial p' that comes equipped with a morphism $e\colon p' \to p$. One can check that p' together with e satisfies the universal property of the equalizer of f and g; see Exercise 4.33. □

[1] If we're being precise, a "(co)equalizer" is an object equipped with a map, but we will use the term to refer to either just the object or just the map when the context is clear.

Exercise 4.33 (Solution here). Complete the proof of Theorem 4.32 as follows:

1. We said that p' comes equipped with a morphism $e\colon p' \to p$; what is it?
2. Show that $e \mathbin{⨟} f = e \mathbin{⨟} g$.
3. Show that e is the equalizer of the pair f, g. ◊

Example 4.34 (Computing general limits in **Poly**). The proof of Theorem 4.32 justifies the following mnemonic for limits in **Poly**:

The positions of a limit are the limit of the positions.
The directions of a limit are the colimit of the directions.

We can make this precise as follows: the limit of a functor $p_-\colon \mathcal{J} \to \mathbf{Poly}$ is the polynomial whose position-set is

$$\left(\lim_{j\in\mathcal{J}} p_j\right)(1) \cong \lim_{j\in\mathcal{J}} p_j(1), \tag{4.35}$$

equipped with a canonical projection π_j to each $p_j(1)$, and whose direction-set for each position i is

$$\left(\lim_{j\in\mathcal{J}} p_j\right)[i] \cong \operatorname*{colim}_{j\in\mathcal{J}^{\mathrm{op}}} p_j[\pi_j(i)]. \tag{4.36}$$

This notation obscures what is occuring on morphisms, but in particular, each morphism $\phi\colon p_j \to p_{j'}$ in the diagram p_- induces an on-positions function $\phi_1\colon p_j(1) \to p_{j'}(1)$ in the diagram whose limit we take in (4.35) and, for every position i of the limit, an on-directions function $\phi^\sharp_{\pi_j(i)}\colon p_{j'}[\pi_{j'}(i)] \to p_j[\pi_j(i)]$ in the diagram whose colimit we take in (4.36). (Note that, by the definition of a limit, $\phi_1(\pi_j(i)) = \pi_{j'}(i)$.)

We have seen (4.35) and (4.36) to be true for products: the position-set of the product is just the product of the original position-sets, while the direction-set at a tuple of the original positions is just the coproduct of the direction-sets at every position in the tuple. We have also just shown (4.35) and (4.36) to be true for equalizers in the proof of Theorem 4.32. It follows from the construction of any limit as an equalizer of products that it is true for arbitrary limits.

Example 4.37 (Pullbacks in **Poly**). Given $q, q', r \in \mathbf{Poly}$ and morphisms $q \xrightarrow{f} r \xleftarrow{f'} q'$, the pullback

$$\begin{tikzcd}
p \arrow[r, "g'"] \arrow[d, "g"'] \arrow[dr, phantom, "\lrcorner", very near start] & q' \arrow[d, "f'"] \\
q \arrow[r, "f"'] & r
\end{tikzcd}$$

is given as follows. The position-set of p is the pullback of the position-sets of q and q'

over that of r in **Set**. Then at each position $(i, i') \in p(1) \subseteq q(1) \times q'(1)$ with $f_1(i) = f_1'(i')$, we take the direction-set $p[(i, i')]$ to be the pushout of the direction-sets $q[i]$ and $q'[i']$ over $r[f_1(i)] = r[f_1'(i')]$ in **Set**. These pullback and pushout squares also given the morphisms g and g' on positions and on directions:

$$\begin{tikzcd}
p(1) \arrow[r, "g_1'"] \arrow[d, "g_1"'] \arrow[dr, phantom, "\lrcorner", very near start] & q'(1) \arrow[d, "f_1'"] & \text{and} & p[(i,i')] \arrow[dr, phantom, "\lrcorner", very near start] & q'[i'] \arrow[l, "(g')^\sharp_{(i,i')}"'] \\
q(1) \arrow[r, "f_1"'] & r(1) & & q[i] \arrow[u, "g^\sharp_{(i,i')}"] & r[f_1(i)] \arrow[l, "f_i^\sharp"] \arrow[u, "(f')^\sharp_{i'}"']
\end{tikzcd} \tag{4.38}$$

Exercise 4.39 (Solution here). Let p be any polynomial.

1. There is a canonical choice of morphism $\eta\colon p \to p(1)$; what is it?
2. Given an element $i \in p(1)$, i.e. a function (or morphism between constant polynomials) $i\colon 1 \to p(1)$, let p_i be the pullback

$$\begin{tikzcd}
p_i \arrow[r, "g"] \arrow[d, "f"'] \arrow[dr, phantom, "\lrcorner", very near start] & p \arrow[d, "\eta"] \\
1 \arrow[r, "i"'] & p(1)
\end{tikzcd}$$

What is p_i? What are the maps $f\colon p_i \to 1$ and $g\colon p_i \to p$? ◊

Exercise 4.40 (Solution here). Let $q := y^2 + y$, $q' := 2y^3 + y^2$, and $r := y + 1$.

1. Choose morphisms $f\colon q \to r$ and $f'\colon q' \to r$ and write them down.
2. Find the pullback of $q \xrightarrow{f} r \xleftarrow{f'} q'$. ◊

Exercise 4.41 (Solution here). An alternative way to prove Theorem 4.32 would have been to show that the equalizer of two natural transformations between polynomial functors in $[\mathbf{Set}, \mathbf{Set}]$ is still a polynomial functor—since the full subcategory inclusion $\mathbf{Poly} \to [\mathbf{Set}, \mathbf{Set}]$ reflects these equalizers, it would follow that **Poly** has equalizers. But we already know what polynomial the equalizer should be from the proof of Theorem 4.32. So in this exercise, we will show that the equalizer of polynomials we found in **Poly** is also the equalizer of those same functors in $[\mathbf{Set}, \mathbf{Set}]$.

Let $f, g\colon p \rightrightarrows q$ be a pair of natural transformations $f, g\colon p \rightrightarrows q$ between polynomial functors p and q, and let $e\colon p' \to p$ be their equalizer in **Poly** that we computed in the proof of Theorem 4.32.

1. Given a set X, show that $e_X\colon p'(X) \to p(X)$ is the equalizer of the X-components $f_X, g_X\colon p(X) \rightrightarrows q(X)$ in **Set**.
2. Deduce that equalizers in **Poly** coincide with equalizers in $[\mathbf{Set}, \mathbf{Set}]$.

3. Conclude that limits in **Poly** coincide with limits in [**Set**, **Set**]. ◊

Theorem 4.42. The category **Poly** has all small colimits.

Proof. A category has all small colimits if and only if it has coproducts and coequalizers, so by Proposition 2.50, it suffices to show that **Poly** has coequalizers.

Let $s, t\colon p \rightrightarrows q$ be two lenses. We construct the coequalizer q' of s and t as follows. The pair of functions $s_1, t_1\colon p(1) \rightrightarrows q(1)$ define a graph $G\colon \boxed{\bullet \rightrightarrows \bullet} \to \textbf{Set}$ with vertices in $q(1)$, edges in $p(1)$, sources indicated by s_1, and targets indicated by t_1. Then the set C of connected components of G is given by the coequalizer $g_1\colon q(1) \to C$ of s_1 and t_1. We define the position-set of q' to be C. Each direction-set of q' will be a limit of a diagram of direction-sets of p and q, but expressing this limit, as we proceed to do, is a bit involved.

For each connected component $c \in C$, we have a connected subgraph $G_c \subseteq G$ with vertices $V_c \coloneqq g_1^{-1}(c)$ and edges $E_c \coloneqq s_1^{-1}(g_1^{-1}(c)) = t_1^{-1}(g_1^{-1}(c))$. Note that $E_c \subseteq p(1)$ and $V_c \subseteq q(1)$, so to each $e \in E_c$ (resp. to each $v \in V_c$) we have an associated direction-set $p[e]$ (resp. $q[v]$).

The category of elements $\int G_c$ has objects $E_c + V_c$ and two kinds of (non-identity) morphisms, $e \to s_1(e)$ and $e \to t_1(e)$, associated to each $e \in E_c$, all pointing from an object in E_c to an object in V_c. There is a functor $F\colon (\int G_c)^{\text{op}} \to \textbf{Set}$ sending every $v \mapsto q[v]$, every $e \mapsto p[e]$, and every morphism to a function between them, namely either $s_e^\sharp\colon q[s_1(e)] \to p[e]$ or $t_e^\sharp\colon q[t_1(e)] \to p[e]$. So we can define $q'[c]$ to be the limit of F in **Set**.

We claim that $q' \coloneqq \sum_{c \in C} y^{q'[c]}$ is the coequalizer of s and t. We leave the complete proof to the interested reader in Exercise 4.43. □

Exercise 4.43 (Solution here). Complete the proof of Theorem 4.42 as follows:

1. Provide a map $g\colon q \to q'$.
2. Show that $s \mathbin{⨾} g = t \mathbin{⨾} g$.
3. Show that g is a coequalizer of the pair s, t. ◊

Example 4.44. Given a diagram in **Poly**, one could either take its (co)limit as a diagram of *polynomial* functors (i.e. its (co)limit in **Poly**) or its (co)limit simply as a diagram of functors (i.e. its (co)limit in [**Set**, **Set**]). We saw in Exercise 4.41 that in the case of limits, these yield the same result. So, too, in the case of coproducts, per Proposition 2.50.

But in the case of general colimits, there are diagrams that yield different results: by the co-Yoneda lemma, *every* functor **Set** → **Set**—even those that are not polynomials—can be written as the colimit of representable functors in [**Set**, **Set**], yet the colimit of the same representables in **Poly** can only be another polynomial.

For a concrete example, consider the two distinct projections $y^2 \to y$, which form the diagram

$$y^2 \rightrightarrows y. \tag{4.45}$$

According to Theorem 4.42, the colimit of (4.45) in **Poly** has the coequalizer of $1 \rightrightarrows 1$, namely 1, as its position-set, and the limit of the diagram $1 \rightrightarrows 2$ consisting of the two inclusions as its sole direction-set. But this latter limit is just 0, so in fact the colimit of (4.45) in **Poly** is the constant functor $1y^0 \cong 1$.

But as functors, by Proposition 2.33, the colimit of (4.45) can be computed pointwise: it is the (nonconstant!) functor

$$X \mapsto \begin{cases} 0 & \text{if } X = 0 \\ 1 & \text{if } X \neq 0 \end{cases}$$

Exercise 4.46 (Solution here). By Proposition 1.22, for any polynomial p, there are canonical maps

$$\epsilon\colon p(1)y \to p \qquad \text{and} \qquad \eta\colon p \to y^{\Gamma(p)}.$$

1. Characterize the behavior of the canonical map $\epsilon\colon p(1)y \to p$.
2. Characterize the behavior of the canonical map $\eta\colon p \to y^{\Gamma(p)}$.
3. Show that the following is a pushout in **Poly**:

$$\begin{tikzcd} p(1)y \arrow[r, "!"] \arrow[d, "\epsilon"'] \arrow[dr, phantom, "\ulcorner", very near end] & y \arrow[d, "!"] \\ p \arrow[r, "\eta"'] & y^{\Gamma(p)} \end{tikzcd} \tag{4.47}$$

◊

Proposition 4.48. For polynomials p, q, the following is a pushout:

$$\begin{tikzcd} p(1)y \otimes q(1)y \arrow[r] \arrow[d] \arrow[dr, phantom, "\ulcorner", very near end] & p(1)y \otimes q \arrow[d] \\ p \otimes q(1)y \arrow[r] & p \otimes q \end{tikzcd}$$

Proof. All the maps shown are identities on positions, so the displayed diagram is the coproduct over all $(i, j) \in p(1) \times q(1)$ of the diagram shown left

$$\begin{tikzcd} y \arrow[r] \arrow[d] \arrow[dr, phantom, "\ulcorner", very near end] & y^{q[j]} \arrow[d] & 1 & q[j] \arrow[l] \\ y^{p[i]} \arrow[r] & y^{p[i] \times q[j]} & p[i] \arrow[u] & p[i] \times q[j] \arrow[l] \arrow[u] \arrow[ul, phantom, "\ulcorner", very near start] \end{tikzcd}$$

where we used $(p\otimes q)[(i,j)] \cong p[i]\times q[j]$. This is the image under the Yoneda embedding of the diagram of sets shown right, which is clearly a pullback. The result follows by Proposition 4.12. □

This means that to give a map $\varphi\colon p\otimes q \to r$, it suffices to give two maps $\varphi_p\colon p\otimes q(1)y \to r$ and $\varphi_q\colon p(1)y\otimes q \to r$ that agree on positions. The map φ_p says how information about q's position is transferred to p, and the map φ_q says how information about p's position is transferred to q.

Corollary 4.49. Suppose we have polynomials $p_1,\ldots,p_n \in \mathbf{Poly}$. Then $p_1\otimes\cdots\otimes p_n$ is isomorphic to the wide pushout

$$\operatorname{colim}\left(\begin{tikzcd}[column sep=small] & p_1(1)y\otimes\cdots\otimes p_n(1)y \arrow[dl] \arrow[dr] & \\ p_1\otimes p_2(1)y\otimes\cdots\otimes p_n(1)y & \cdots & p_1(1)y\otimes\cdots\otimes p_{n-1}(1)y\otimes p_n \end{tikzcd}\right)$$

Proof. We proceed by induction on $n\in\mathbb{N}$. When $n=0$, the wide pushout has no legs and the empty parallel product is y, so the result holds. If the result holds for n, then it holds for $n+1$ by Proposition 4.48. □

4.5 Vertical-cartesian factorization

Aside from epi-mono factorization, there is another factorization system on **Poly** that will show up frequently.

Definition 4.50 (Vertical, cartesian)**.** Let $f\colon p\to q$ be a lens. It is called *vertical* if $f_1\colon p(1)\to q(1)$ is an isomorphism. It is called *cartesian* if, for each $i\in p(1)$, the function $f_i^\sharp\colon q[f(i)]\to p[i]$ is an isomorphism.

Proposition 4.51. Vertical and cartesian morphisms form a factorization system of **Poly**.

Proof. It is easy to check that isomorphisms are both vertical and cartesian, and that vertical and cartesian morphisms are each closed under composition. It remains to show that every morphism in **Poly** can be uniquely (up to unique isomorphism) factored as a vertical morphism composed with a cartesian morphism.

Recall from (2.55) that a morphism in **Poly** can be written as to the left; we can thus rewrite it as to the right:

$$\begin{tikzcd} p(1) \arrow[rr, "f_1"] \arrow[dr, bend right, "{p[-]}"'] & \arrow[d, phantom, "\overset{f^\sharp}{\Leftarrow}"] & q(1) \arrow[dl, bend left, "{q[-]}"] \\ & \mathbf{Set} & \end{tikzcd} \qquad \begin{tikzcd} p(1) \arrow[r, equal, "\overset{f^\sharp}{\Leftarrow}"'] \arrow[dr, bend right, "{p[-]}"'] & p(1) \arrow[r, "f_1"] \arrow[d, "{q[f_1(-)]}"] & q(1) \arrow[dl, bend left, "{q[-]}"] \\ & \mathbf{Set} & \end{tikzcd}$$

The intermediary object $\sum_{i \in p(1)} y^{q[f_1(i)]}$ is clearly unique up to unique isomorphism. □

Proposition 4.52. Vertical morphisms satisfy 2-out-of-3: given $p \xrightarrow{f} q \xrightarrow{g} r$ with $h = f \mathbin{⨾} g$, if any two of f, g, h are vertical, then so is the third.

If g is cartesian, then h is cartesian if and only if f is cartesian.

Proof. Given $h = f \mathbin{⨾} g$, we have that $h_1 = f_1 \mathbin{⨾} g_1$. Since isomorphisms satisfy 2-out-of-3, it follows that vertical morphisms satisfy 2-out-of-3 as well.

Now assume g is cartesian. On directions, $h = f \mathbin{⨾} g$ implies that for every $i \in p(1)$, we have $h_i^\sharp = g_{f_1(i)}^\sharp \mathbin{⨾} f_i^\sharp$. Since $g_{f_1(i)}^\sharp$ is an isomorphism, it follows that every $h_i^\sharp$ is an isomorphism if and only if every $f_i^\sharp$ is an isomorphism, so h is cartesian if and only if f is cartesian. □

Exercise 4.53 (Solution here). Give an example of polynomials p, q, r and maps $p \xrightarrow{f} q \xrightarrow{g} r$ such that f and $f \mathbin{⨾} g$ are cartesian but g is not. ◊

Here is an alternative characterization of a cartesian morphism in **Poly**. Recall from Exercise 2.65 that for any polynomial p, there is a corresponding function $\pi_p \colon \dot{p}(1) \to p(1)$, i.e. the set of all directions mapping to the set of positions. A lens $(f_1, f^\sharp) \colon p \to q$ can then be described as a function $f_1 \colon p(1) \to q(1)$ along with a function $f^\sharp$ that makes the following diagram in **Set** commute:

$$\begin{tikzcd}
\dot{p}(1) \arrow[d, "\pi_p"'] & \bullet \arrow[l, "f^\sharp"'] \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \dot{q}(1) \arrow[d, "\pi_q"] \\
p(1) \arrow[r, equal] & p(1) \arrow[r, "f_1"'] & q(1)
\end{tikzcd} \tag{4.54}$$

Here, the pullback denoted by the dot • is the set of pairs comprised of a p-position i and a $q[f_1(i)]$-direction e. The function $f^\sharp$ sends each such pair to a direction $f_i^\sharp(e)$ of p, and the commutativity of the left square implies that $f_i^\sharp(e)$ is specifically a $p[i]$-direction. So $f_i^\sharp$ is indeed our familiar on-directions function $q[f_1(i)] \to p[i]$, and $f^\sharp$ is just the sum of all these on-directions functions over $i \in p(1)$.

Exercise 4.55 (Solution here). Show that a morphism $f \colon p \to q$ in **Poly** is cartesian if and only if the square on the left hand side of (4.54) is also a pullback:

$$\begin{tikzcd}
\dot{p}(1) \arrow[d, "\pi_p"'] & \bullet \arrow[l, "f^\sharp"'] \arrow[r] \arrow[d] \arrow[dl, phantom, "\llcorner", very near start] \arrow[dr, phantom, "\lrcorner", very near start] & \dot{q}(1) \arrow[d, "\pi_q"] \\
p(1) \arrow[r, equal] & p(1) \arrow[r, "f_1"'] & q(1)
\end{tikzcd}$$

◊

Exercise 4.56 (Solution here). Is the pushout of a cartesian map always cartesian? ◊

Why do we use the word *cartesian* to describe cartesian morphisms? It turns out that, as natural transformations, cartesian morphisms are precisely what are known as cartesian natural transformations.

Definition 4.57 (Cartesian natural transformation)**.** A *cartesian natural transformation* is a natural transformation whose naturality squares are all pullbacks. That is, given categories $\mathcal{C}, \mathcal{D}$, functors F, G, and natural transformation α, we say that α is *cartesian* if for all morphisms $h \colon c \to c'$ in $\mathcal{C}$,

$$\begin{tikzcd}
Fc \arrow[r, "\alpha_c"] \arrow[d, "Fh"'] \arrow[dr, phantom, very near start, "\lrcorner"] & Gc \arrow[d, "Gh"] \\
Fd \arrow[r, "\alpha_d"'] & Gd
\end{tikzcd}$$

is a pullback.

Proposition 4.58. Let $f \colon p \to q$ be a morphism in **Poly**. The following are equivalent:

1. viewed as a lens, f is a cartesian morphism in the sense of Definition 4.50: for each $i \in p(1)$, the on-directions function $f_i^\sharp$ is a bijection;
2. the square on the left hand side of (4.54) is also a pullback:

$$\begin{tikzcd}
\dot{p}(1) \arrow[d, "\pi_p"'] & \bullet \arrow[l, "f^\sharp"'] \arrow[r] \arrow[d] \arrow[dl, phantom, very near start, "\llcorner"] \arrow[dr, phantom, very near start, "\lrcorner"] & \dot{q}(1) \arrow[d, "\pi_q"] \\
p(1) \arrow[r, equal] & p(1) \arrow[r, "f_1"'] & q(1)
\end{tikzcd}$$

3. viewed as a natural transformation, f is cartesian in the sense of Definition 4.57: for any sets A, B and function $h \colon A \to B$, the naturality square

$$\begin{tikzcd}
p(A) \arrow[r, "f_A"] \arrow[d, "p(h)"'] \arrow[dr, phantom, very near start, "\lrcorner"] & q(A) \arrow[d, "q(h)"] \\
p(B) \arrow[r, "f_B"'] & q(B)
\end{tikzcd} \tag{4.59}$$

is a pullback.

Proof. We already showed that the first two are equivalent in Exercise 4.55, and we will complete this proof in Exercise 4.60. □

Exercise 4.60 (Solution here). In this exercise, you will complete the proof of Proposition 4.58.

First, we will show that $1 \Rightarrow 3$. In the following, let $f\colon p \to q$ be a cartesian morphism in **Poly** and $h\colon A \to B$ be a function.

1. Using Proposition 2.70 to translate f from a morphism in **Poly** to a natural transformation and Proposition 2.46 to interpret $q(h)$, characterize the pullback of $p(B) \xrightarrow{f_B} q(B) \xleftarrow{q(h)} q(A)$ in **Set**.
2. Show that this pullback coincides with the naturality square (4.59), hence proving $1 \Rightarrow 3$.

Next, we show that $3 \Rightarrow 1$. In the following, let $f\colon p \to q$ be a morphism in **Poly** that is cartesian when viewed as a natural transformation, so that (4.59) is a pullback for any function $h\colon A \to B$. Also fix $i \in p(1)$.

3. Show that the diagram

$$\begin{tikzcd}
1 \arrow[r, "{(f_1(i), \mathrm{id}_{q[f_1(i)]})}"] \arrow[d, "{(i, \mathrm{id}_{p[i]})}"'] \arrow[dr, phantom, "\lrcorner", very near start] & q(q[f_1(i)]) \arrow[d, "q(f_i^\sharp)"] \\
p(p[i]) \arrow[r, "f_{p[i]}"'] & q(p[i])
\end{tikzcd} \tag{4.61}$$

commutes. Hint: Use Proposition 2.46, Proposition 2.70, and/or Corollary 2.72.

4. Apply the universal property of the pullback (4.59) to the diagram (4.61) above to exhibit an element of $p(q[f_1(i)])$. Conclude from the existence of this element that $f_i^\sharp$ is an isomorphism, hence proving $3 \Rightarrow 1$. ◊

Proposition 4.62. The monoidal structures $+$, $\times$, and $\otimes$ preserve both vertical and cartesian morphisms.

Proof. Suppose that $f\colon p \to p'$ and $g\colon q \to q'$ are vertical, so that the on-positions functions f_1 and g_1 are isomorphisms.

We can obtain the on-positions function of a morphism by passing it through the functor $\mathbf{Poly} \xrightarrow{p(1)} \mathbf{Set}$ from Theorem 4.4. As this functor is both a left adjoint and a right adjoint, it preserves both sums and products, so $(f+g)_1 = f_1 + g_1$ and $(f \times g)_1 = f_1 \times g_1$. Hence $f + g$ and $f \times g$ are both vertical. On-positions, the behavior of $\otimes$ is identical to the behavior of $\times$, so $f \otimes g$ must be vertical as well.

Now suppose that $f\colon p \to p'$ and $g\colon q \to q'$ are cartesian.

A position of $p + q$ is a position $i \in p(1)$ or a position $j \in q(1)$, and the map $(f+g)^\sharp$ at that position is either $f_i^\sharp$ or $g_j^\sharp$; either way it is an isomorphism, so $f + g$ is cartesian.

A position of $p \times q$ (resp. of $p \otimes q$) is a pair $(i, j) \in p(1) \times q(1)$. The morphism $(f \times g)^\sharp_{(i,j)}$ (resp. $(f \otimes g)^\sharp_{(i,j)}$) is $f_i^\sharp + g_j^\sharp$ (resp. $f_i^\sharp \times g_j^\sharp$) which is again an isomorphism if $f_i^\sharp$ and $g_j^\sharp$ are. Hence $f \times g$ (resp. $f \otimes g$) is cartesian, completing the proof. □

Proposition 4.63. Pullbacks preserve vertical (resp. cartesian) morphisms. In other words, if $f\colon p \to q$ is a morphism and $g\colon q' \to q$ a vertical (resp. cartesian) morphism, then the pullback g' of g along p

$$\begin{tikzcd}
p \times_q q' \arrow[r] \arrow[d, "g'"'] \arrow[dr, phantom, "\lrcorner", very near start] & q' \arrow[d, "g"] \\
p \arrow[r, "f"'] & q
\end{tikzcd}$$

is vertical (resp. cartesian).

Proof. This follows from Example 4.37, since the pullback (resp. pushout) of an isomorphism is an isomorphism. □

4.6 Monoidal $*$-bifibration over Set

We will see that the functor $p \mapsto p(1)$ has special properties making it what [Shu08] refers to as a *monoidal $*$-bifibration*. This means that **Set** acts as a sort of remote controller on the category **Poly**, grabbing every polynomial by its positions and pushing or pulling it this way and that.

For example, suppose one has a set A and a function $f\colon A \to p(1)$, which we can also think of as a cartesian morphism between constant polynomials. Then we get a new polynomial f^*p with positions A, as follows. It is given by a pullback

$$\begin{tikzcd}
f^*p \arrow[r, "\text{cart}"] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & p \arrow[d, "\eta_p"] \\
A \arrow[r, "f"'] & p(1)
\end{tikzcd} \tag{4.64}$$

Here η_p is the unit of the adjunction $\begin{tikzcd} \mathbf{Set} \arrow[r, shift left, "A"] \arrow[r, phantom, "\Leftarrow"] & \mathbf{Poly} \arrow[l, shift left, "p(1)"] \end{tikzcd}$; it is a vertical morphism. We could evaluate this pullback using Example 4.37. Alternatively, we can use Proposition 4.63 to deduce that the top map $f^*p \to p$ (which we presciently labeled cart) is cartesian like f and that the left map $f^*p \to A$ is vertical like η_p. Furthermore, $\text{cart}_1 = f$. Hence

$$f^*p \cong \sum_{a \in A} y^{p[f(a)]}.$$

We'll see this as part of a bigger picture in Proposition 4.70 and Theorem 4.71, but first we need the following definition and result about cartesian morphisms.

For any $p \in \mathbf{Poly}$, let $\mathbf{Poly}/p$ denote the slice category, i.e. the category whose objects are maps to p and whose morphisms are commutative triangles.

Definition 4.65. Given a category $\mathcal{C}$ with objects c, d and morphism $f\colon c \to d$ such that all pullbacks along f exist in $\mathcal{C}$, we say that f is *exponentiable* if the functor $f^*\colon \mathcal{C}/d \to \mathcal{C}/c$ given by pulling back along f is a left adjoint.

Theorem 4.66. Cartesian morphisms in **Poly** are exponentiable. That is, if $f\colon p \to q$ is cartesian, then the functor $f^*\colon \mathbf{Poly}/q \to \mathbf{Poly}/p$ given by pulling back along f is a left adjoint:

$$\begin{tikzcd}
\mathbf{Poly}/p \arrow[r, shift right=1.5ex, "f_*"'] \arrow[r, phantom, "\Leftarrow"] & \mathbf{Poly}/q \arrow[l, shift right=1.5ex, "f^*"']
\end{tikzcd}$$

Proof. Fix $e\colon p' \to p$ and $g\colon q' \to q$.

$$\begin{tikzcd}
p' \arrow[d, "e"'] & q' \arrow[d, "g"] \\
p \arrow[r, "f"'] & q
\end{tikzcd}$$

We need to define a functor $f_*\colon \mathbf{Poly}/p \to \mathbf{Poly}/q$ and prove the analogous isomorphism establishing it as right adjoint to f^*. We first establish some notation. Given a set Q and sets $(P'_i)_{i\in I}$, each equipped with a map $Q \to P'_i$, let $Q/\sum_{i\in I} P'_i$ denote the coproduct in $Q/\mathbf{Set}$, or equivalently the wide pushout of sets P'_i with apex Q. Then we give the following formula for f_*p', which we write in larger font for clarity:

$$f_*p' \coloneqq \sum_{j\in q(1)} \sum_{i'\in \prod\limits_{i\in f_1^{-1}(j)} e_1^{-1}(i)} y^{q[j]/\sum_{i\in f_1^{-1}(j)} p'[i'(i)]} \tag{4.67}$$

Again, $q[j]/\sum_{i\in f_1^{-1}(j)} p'[i'(i)]$ is the coproduct of the $p'[i'(i)]$, taken in $q[j]/\mathbf{Set}$. Since $p[i] \cong q[f(i)]$ for any $i \in p(1)$ by the cartesian assumption on f, we have the following chain of natural isomorphisms

$$\begin{aligned}
(\mathbf{Poly}/p)(f^*q', p') &\cong \prod_{i\in p(1)} \prod_{\{j'\in q'(1)\,|\,g_1(j')=f_1(i)\}} \sum_{\{i'\in p'(1)\,|\,e_1(i')=i\}} (p[i]/\mathbf{Set})(p'[i'], p[i] +_{q[f(i)]} q'[j']) \\
&\cong \prod_{i\in p(1)} \prod_{\{j'\in q'(1)\,|\,g_1(j')=f_1(i)\}} \sum_{\{i'\in p'(1)\,|\,e_1(i')=i\}} (q[f(i)]/\mathbf{Set})(p'[i'], q'[j']) \\
&\cong \prod_{j\in q(1)} \prod_{\{j'\in q'(1)\,|\,g_1(j')=j\}} \prod_{\{i\in p(1)\,|\,f_1(i)=j\}} \sum_{\{i'\in p'(1)\,|\,e_1(i')=i\}} (q[j]/\mathbf{Set})(p'[i'], q'[j']) \\
&\cong \prod_{j\in q(1)} \prod_{\{j'\in q'(1)\,|\,g_1(j')=j\}} \sum_{i'\in \prod_{i\in f_1^{-1}(j)} e_1^{-1}(i)} \prod_{i\in f_1^{-1}(j)} (q[j]/\mathbf{Set})(p'[i'(i)], q'[j']) \\
&\cong \prod_{j\in q(1)} \prod_{\{j'\in q'(1)\,|\,g_1(j')=j\}} \sum_{i'\in \prod_{i\in f_1^{-1}(j)} e_1^{-1}(i)} (q[j]/\mathbf{Set})\Big(\sum_{i\in f_1^{-1}(j)} p'[i'(i)], q'[j']\Big)
\end{aligned}$$

$$\cong (\mathbf{Poly}/q)(q', f_* p')$$

□

Example 4.68. Let $p := 2y^2$, $q := y^2 + y^0$, and $f : p \to q$ the unique cartesian morphism between them. Then for any $e : p' \to p$ over p, (4.67) provides the following description for the pushforward $f_* p'$.

Over the $j = 2$ position, $f_1^{-1}(2) = 0$ and $q[2] = 0$, so $\prod_{i \in f_1^{-1}(2)} e_1^{-1}(i)$ is an empty product and $q[2]/\sum_{i \in f_1^{-1}(2)} p'[i'(i)]$ is an empty pushout. Hence the corresponding summand of (4.67) is simply $y^0 \cong 1$.

Over the $j = 1$ position, $f_1^{-1}(1) = 2$ and $q[1] = p[1] = p[2] = 2$, so $\prod_{i' \in f_1^{-1}(1)} e_1^{-1}(i) \cong e_1^{-1}(1) \times e_1^{-1}(2)$. For $i' \in e_1^{-1}(1) \times e_1^{-1}(2)$, we have that $q[1]/\sum_{i \in f_1^{-1}(2)} p'[i'(i)] \cong X_{i'}$ in the following pushout square:

$$\begin{tikzcd}
X_{i'} \arrow[dr, phantom, "\lrcorner", very near start] & p'[i'(2)] \arrow[l] \\
p'[i'(1)] \arrow[u] & 2 \arrow[l, "e^\sharp_{i'(1)}"'] \arrow[u, "e^\sharp_{i'(2)}"']
\end{tikzcd}$$

Then in sum we have

$$f_* p' \cong \left(\sum_{i' \in e_1^{-1}(1) \times e_2^{-1}(2)} y^{X_{i'}} \right) + 1.$$

Exercise 4.69 (Solution here). Prove that the unique map $f : y \to 1$ is exponentiable. ◊

For any set A, let $A.\mathbf{Poly}$ denote the category whose objects are polynomials p equipped with an isomorphism $A \cong p(1)$, and whose morphisms are lenses respecting the isomorphisms with A.

Proposition 4.70 (Base change)**.** For any function $f : A \to B$, pullback f^* along f induces a functor $B.\mathbf{Poly} \to A.\mathbf{Poly}$, which we also denote f^*.

Proof. This follows from (4.38) with $q := A$ and $r := B$, since pullback of an iso is an iso. □

Theorem 4.71. For any function $f\colon A \to B$, the pullback functor f^* has both a left and a right adjoint

$$\begin{tikzcd}
A.\mathbf{Poly} \arrow[r, shift left=5, "f_!"] \arrow[r, phantom, shift left=2.5, "\Rightarrow"] \arrow[r, phantom, shift right=2.5, "\Leftarrow"] \arrow[r, shift right=5, "f_*"'] & B.\mathbf{Poly} \arrow[l, "f^*" description]
\end{tikzcd} \tag{4.72}$$

Moreover $\otimes$ preserves the op-cartesian arrows, making this a monoidal ∗-bifibration in the sense of [Shu08, Definition 12.1].

Proof. Let p be a polynomial with $p(1) \cong A$. Then the formula for $f_!p$ and f_*p are given as follows:

$$f_!p \cong \sum_{b\in B} y^{\left(\prod_{a\mapsto b} p[a]\right)} \qquad \text{and} \qquad f_*p \cong \sum_{b\in B} y^{\left(\sum_{a\mapsto b} p[a]\right)} \tag{4.73}$$

It may at first be counterintuitive that the left adjoint $f_!$ involves a product and the right adjoint f_* involves a sum. The reason for this comes from **??**, namely that **Poly** is equivalent to the Grothendieck construction applied to the functor $\mathbf{Set}^{\mathrm{op}} \to \mathbf{Cat}$ sending each set A to the category $(\mathbf{Set}/A)^{\mathrm{op}}$. The fact that functions $f\colon A \to B$ induces an adjoint triple between $\mathbf{Set}/A$ and $\mathbf{Set}/B$, and hence between $(\mathbf{Set}/A)^{\mathrm{op}}$ and $(\mathbf{Set}/B)^{\mathrm{op}}$ explains the variance in (4.73) and simultaneously establishes the adjoint triple (4.72).

The functor $p \mapsto p(1)$ is strong monoidal with respect to $\otimes$ and strict monoidal if we choose the lens construction as our model of **Poly**. By Proposition 4.62, the monoidal product $\otimes$ preserves cartesian morphisms; thus we will have established the desired monoidal ∗-bifibration in the sense of [Shu08, Definition 12.1] as soon as we know that $\otimes$ preserves op-cartesian morphisms.

Given f and p as above, the op-cartesian morphism is the morphism $p \to f_!p$ obtained as the composite $p \to f^*f_!p \to f_!p$ where the first map is the unit of the $(f_!, f^*)$ adjunction and the second is the cartesian morphism for $f_!p$. On positions $p \to f_!p$ acts as f, and on directions it is given by projection.

If $f\colon p(1) \to B$ and $f'\colon p'(1) \to B'$ are functions then we have

$$\begin{aligned}
f_!(p) \otimes f'_!(p') &\cong \sum_{b\in B}\sum_{b'\in B'} y^{\left(\prod_{a\mapsto b} p[a]\right)\times\left(\prod_{a'\mapsto b'} p'[a']\right)} \\
&\cong \sum_{(b,b')\in B\times B'} y^{\left(\prod_{(a,a')\mapsto(b,b')} p[a]\times p[b]\right)} \\
&\cong (f_! \otimes f'_!)(p \otimes p')
\end{aligned}$$

and the op-cartesian morphisms are clearly preserved since projections in the second line match with projections in the first. □

4.7 Summary and further reading

...

See work by Gambino and Kock, Joyal, Paul Taylor, Michael Abbott and Neil Ghani (containers), ...

4.8 Exercise solutions

Solution to Exercise 4.1.

Here $A, B, A', B' \in \mathbf{Set}$.

1. We determine whether various classes of polynomials are closed under addition.
   a) Constant polynomials are closed under addition: given constants A, B, their sum $A + B$ is also a constant polynomial.
   b) Linear polynomials are closed under addition: given linear polynomials Ay, By, their sum $Ay + By \cong (A + B)y$ is also a linear polynomial.
   c) Representable polynomials are *not* closed under addition: for example, y is a representable polynomial, but the sum of y with itself, $2y$, is not.
   d) Monomials are *not* closed under addition: for example, y and $2y^3$ are monomials, but their sum $y + 2y^3$ is not.
2. We determine whether various classes of polynomials are closed under multiplication. The results below follow from Exercise 2.86 #1.
   a) Constant polynomials are closed under multiplication: given constants A, B, their product AB is also a constant polynomial.
   b) Linear polynomials are *not* closed under multiplication: for example, y and $2y$ are linear polynomials, but their product $2y^2$ is not.
   c) Representable polynomials are closed under multiplication: given representables y^A, y^B, their product y^{A+B} is also a representable polynomial.
   d) Monomials are closed under multiplication: given monomials $By^A, B'y^{A'}$, their product $BB'y^{A+A'}$ is also a monomial.
3. We determine whether various classes of polynomials are closed under taking parallel products. The results below follow from Exercise 2.91 #1.
   a) Constant polynomials are closed under taking parallel products: given constants A, B, their parallel product AB is also a constant polynomial.
   b) Linear polynomials are closed under taking parallel products: given linear polynomials Ay, By, their parallel product ABy is also a linear polynomial.
   c) Representable polynomials are closed under taking parallel products: given representables y^A, y^B, their parallel product y^{AB} is also a representable polynomial.
   d) Monomials are closed under taking parallel products: given monomials $By^A, B'y^{A'}$, their parallel product $BB'y^{AA'}$ is also a monomial.
4. We determine whether various classes of polynomials are closed under composition. (Recall that we can think of computing the composite $p \circ q$ of $p, q \in \mathbf{Poly}$ as replacing each appearance of y in p with q.)
   a) Constant polynomials are closed under composition: given constants A, B, their composite $A \circ B \cong A$ is also a constant polynomial.
   b) Linear polynomials are closed under composition: given linear polynomials Ay, By, their composite $Ay \circ By \cong A(By) \cong ABy$ is also a linear polynomial.
   c) Representable polynomials are closed under composition: given representables y^A, y^B, their composite $y^A \circ y^B \cong (y^B)^A \cong y^{BA}$ is also a representable polynomial.
   d) Monomials are closed under taking parallel products: given monomials $By^A, B'y^{A'}$, their composite $By^A \circ B'y^{A'} \cong B(B'y^{A'})^A \cong BB'^Ay^{A'A}$ is also a monomial.

Solution to Exercise 4.6.

We complete the proof of Theorem 4.4 by exhibiting three natural isomorphisms, all special cases of (2.54), as follows.

1. By (2.54), we have the natural isomorphism
$$\mathbf{Poly}(A, p) \cong \prod_{a \in A} \sum_{i \in p(1)} 0^{p[i]}.$$
As $0^{p[i]}$ is 1 if $p[i] \cong 0$ and 0 otherwise, it follows that
$$\mathbf{Poly}(A, p) \cong \prod_{a \in A} \{i \in p(1) \mid p[i] \cong 0\} \cong \prod_{a \in A} p(0) \cong \mathbf{Set}(A, p(0)).$$
2. By (2.54), we have the natural isomorphism
$$\begin{aligned}\mathbf{Poly}(p, A) &\cong \prod_{i \in p(1)} \sum_{a \in A} p[i]^0 \\ &\cong \prod_{i \in p(1)} \sum_{a \in A} 1 \\ &\cong \prod_{i \in p(1)} A \\ &\cong \mathbf{Set}(p(1), A).\end{aligned}$$
3. By (2.54), we have the natural isomorphism
$$\begin{aligned}\mathbf{Poly}(Ay, p) &\cong \prod_{a \in A} \sum_{i \in p(1)} 1^{p[i]} \\ &\cong \prod_{a \in A} \sum_{i \in p(1)} 1 \\ &\cong \prod_{a \in A} p(1) \\ &\cong \mathbf{Set}(A, p(1)).\end{aligned}$$

Solution to Exercise 4.7.

Given $p \in \mathbf{Poly}$, we wish to show that $p(1)$ is in bijection with the set of functions $y \to p$. In fact, this follows directly from the Yoneda lemma, but we can also invoke the isomorphism from Exercise 4.6 #3 with $A := 1$ to observe that
$$p(1) \cong \mathbf{Set}(1, p(1)) \cong \mathbf{Poly}(y, p).$$

Solution to Exercise 4.11.

To prove Corollary 4.10, it suffices to exhibit a natural isomorphism
$$\mathbf{Poly}(Ay^B, q) \cong \mathbf{Set}(A, q(B)).$$
Replacing p with y^B in (4.9) from Proposition 4.8, we obtain the natural isomorphism
$$\mathbf{Poly}(Ay^B, q) \cong \mathbf{Set}(A, \mathbf{Poly}(y^B, q)).$$
By the Yoneda lemma, $\mathbf{Poly}(y^B, q)$ is naturally isomorphic to $q(B)$, yielding the desired result.

Solution to Exercise 4.16.

We have the following chain of natural isomorphisms:
$$\begin{aligned}\Gamma(p \otimes q) &= \mathbf{Poly}(p \otimes q, y) && (3.39) \\ &\cong \mathbf{Poly}(p, [q, y]) && (3.88) \\ &\cong \mathbf{Poly}(p, \Gamma(q) y^{q(1)}) && (3.84) \\ &\cong \mathbf{Set}\big(p(1), \Gamma(q)\big) \times \mathbf{Set}\big(q(1), \Gamma(p)\big). && (4.15)\end{aligned}$$

Solution to Exercise 4.19.

We are given a monomorphism $f\colon 12y^{12} \to \mathbb{N}y^{\mathbb{N}}$ from Example 4.18. Let $g\colon \mathbb{N}y^{\mathbb{N}} \to p$ be a dynamical system with return function $g_1\colon \mathbb{N} \to p(1)$ and update functions $g^\sharp_n\colon p[g_1(n)] \to \mathbb{N}$ for each state $n \in \mathbb{N}$. Then the new composite dynamical system $h \coloneqq f \mathbin{⨾} g$ has a return function $h_1\colon 12 \to p(1)$ which sends each state $i \in 12$ to the output $h_1(i) = g_1(f_1(i)) = g_1(i)$, the same output that the original system returned in the state $i \in \mathbb{N}$. Meanwhile, the update function for each state $i \in 12$ is a function $h^\sharp_i\colon p[g_1(i)] \to 12$ which, given an input $d \in p[g_1(i)]$, updates the state from i to $h^\sharp_i(d) = f^\sharp_{g_1(i)}(g^\sharp_i(d)) = g^\sharp_i(d) \mod 12$, which is where the original system would have taken the same state to, but reduced modulo 12. In other words, the new system behaves like the old system but with only the states in $12 \subseteq \mathbb{N}$ retained, and on any input that would have caused the old system to move to a state outside of 12, the new system moves to the equivalent state (modulo 12) within 12 instead.

Solution to Exercise 4.21.

Given $p \in \textbf{Poly}$ and a map $f\colon p \to y$, we will use Proposition 4.20 to show that either f is an epimorphism or $p = 0$. First, note that $f_1\colon p(1) \to 1$ must be an epimorphism unless $p(1) \cong 0$, in which case $p = 0$. Next, note that the induced function

$$f^\flat\colon 1 \to \prod_{i \in p(1)} p[i]$$

from (2.62) must be a monomorphism. So it follows from Proposition 4.20 that either f is an epimorphism or $p = 0$.

Solution to Exercise 4.22.

Given sets A and B, by Proposition 4.20, a morphism $f\colon y^A + y^B \to y^{AB}$ is an epimorphism if its on-positions function $f_1\colon 2 \to 1$ is an epimorphism (which must be true) and if the induced function

$$f^\flat\colon AB \to \prod_{i \in 2} (y^A + y^B)[i] \cong AB$$

is a monomorphism. If we take the on-directions functions $AB \to A$ and $AB \to B$ of f to be the canonical projections, then the induced function $f^\flat\colon AB \to AB$ would be the identity, which is indeed a monomorphism. So f would be an epimorphism.

Solution to Exercise 4.23.

Let $f\colon p \to q$ be a morphism in **Poly** that is both a monomorphism and an epimorphism. We claim that f is an isomorphism. By Proposition 4.17 and Proposition 4.20, the on-positions function $f_1\colon p(1) \to q(1)$ is both a monomorphism and an epimorphism, so it is an isomorphism. Meanwhile, Proposition 4.20 says that, for each $j \in q(1)$, the induced function

$$f^\flat_j\colon q[j] \to \prod_{\substack{i \in p(1),\\ f_1(i) = j}} p[i]$$

is a monomorphism. As f_1 is an isomorphism, it follows that for each $i \in p(1)$, the function

$$f^\flat_{f_1(i)}\colon q[f_1(i)] \to p[i]$$

is a monomorphism. But this is just the on-directions function $f^\sharp_i$ of f. From Proposition 4.17, we also know that $f^\sharp_i$ is an epimorphism. It follows that every on-directions function of f is an isomorphism. Hence f itself is an isomorphism.

Solution to Exercise 4.29.

We use (4.27) to compute various exponentials. Here $p \in \textbf{Poly}$ and $A, B \in \textbf{Set}$.

1. We have that p^0 is an empty product, so $p^0 \cong 1$ as expected.

2. We have that $p^1 \cong p \circ (y + 0) \cong p$, as expected.
3. We have that $1^p \cong \prod_{i \in p(1)} 1 \circ (y + p[i]) \cong 1$, as expected.
4. We have that $A^p \cong \prod_{i \in p(1)} A \circ (y + p[i]) \cong A^{p(1)}$.
5. We have that $y^y \cong y \circ (y + 1) \cong y + 1$.
6. We have that $y^{4y} \cong \prod_{j \in 4} y \circ (y + 1) \cong (y + 1)^4 \cong y^4 + 4y^3 + 6y^2 + 4y + 1$.
7. We have that $(y^A)^{y^B} \cong (y^A) \circ (y + B) \cong (y + B)^A \cong \sum_{f\colon A \to 2} B^{f^{-1}(1)} y^{f^{-1}(2)}$.

Solution to Exercise 4.31.

By Theorem 4.30, there is a natural isomorphism

$$\mathbf{Poly}(p^q, p^q) \cong \mathbf{Poly}(p^q \times q, p).$$

Under this isomorphism, there exists a map $\text{eval}\colon p^q \times q \to p$ corresponding to the identity map on p^q. The map eval is the canonical evaluation map.

Solution to Exercise 4.33.

1. The morphism $e\colon p' \to p$ can be characterized as follows. The on-positions function $e_1\colon p'(1) \to p(1)$ is the equalizer of $f_1, g_1\colon p(1) \rightrightarrows q(1)$ in **Set**. In particular, e_1 is the canonical inclusion that sends each element of $p'(1)$ to the same element in $p(1)$. Then for each $i \in p'(1)$, the on-directions function $e_i^\sharp\colon p[i] \to p'[i]$ is the coequalizer of $f_i^\sharp, g_i^\sharp\colon q[f_1(i)] \rightrightarrows p[i]$ in **Set**.
2. To show that $e \fatsemi f = e \fatsemi g$, it suffices to show that both sides are equal on positions and on directions. On positions, e_1 is defined to be the equalizer of f_1 and g_1, so $e_1 \fatsemi f_1 = e_1 \fatsemi g_1$. Then for each $i \in p'(1)$, the on-directions function $e_i^\sharp$ is defined to be the coequalizer of $f_i^\sharp$ and $g_i^\sharp$, so $f_i^\sharp \fatsemi e_i^\sharp = g_i^\sharp \fatsemi e_i^\sharp$.
3. To show that e is the equalizer of f and g, it suffices to show that for any $r \in \mathbf{Poly}$ and map $a\colon r \to p$ satisfying $a \fatsemi f = a \fatsemi g$, there exists a unique map $h\colon r \to p'$ for which $a = h \fatsemi e$, so that the following diagram commutes.

$$\begin{tikzcd}
p' \arrow[r, "e"] & p \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & q \\
r \arrow[u, dashed, "h"] \arrow[ur, "a"'] & &
\end{tikzcd}$$

In order for $a = h \fatsemi e$ to hold, we must have $a_1 = h_1 \fatsemi e_1$ on positions. But we have that $a_1 \fatsemi f_1 = a_1 \fatsemi g_1$, so by the universal property of $p'(1)$ and the map e_1 as the equalizer of f_1 and g_1 in **Set**, there exists a unique h_1 for which $a_1 = h_1 \fatsemi e_1$. Hence h is uniquely characterized on positions. In particular, it must send each $k \in r(1)$ to $a_1(k) \in p'(1)$.

Then for $a = h \fatsemi e$ to hold on directions, we must have that $a_k^\sharp = e_{a_1(k)}^\sharp \fatsemi h_k^\sharp$ for each $k \in r(1)$. But we have that $f_{a_1(k)}^\sharp \fatsemi a_{a_1(k)}^\sharp = g_{a_1(k)}^\sharp \fatsemi a_{a_1(k)}^\sharp$, so by the universal property of $p'[a_1(k)]$ and the map $e_{a_1(k)}^\sharp$ as the coequalizer of $f_{a_1(k)}^\sharp$ and $g_{a_1(k)}^\sharp$ in **Set**, there exists a unique $h_k^\sharp$ for which $a_k^\sharp = e_{a_1(k)}^\sharp \fatsemi h_k^\sharp$, so that the diagram below commutes.

$$\begin{tikzcd}
p'[a_1(k)] \arrow[d, dashed, "h_k^\sharp"'] & p[a_1(k)] \arrow[l, "e_{a_1(k)}^\sharp"'] \arrow[dl, "a_k^\sharp"] & q[f_1(a_1(k))] \arrow[l, shift right, "f_{a_1(k)}^\sharp"'] \arrow[l, shift left, "g_{a_1(k)}^\sharp"] \\
r[k] & &
\end{tikzcd}$$

Hence h is also uniquely characterized on directions, so it is unique overall. Moreover, we have shown that we can define h on positions so that $a_1 = h_1 \fatsemi e_1$, and that we can define h on directions such that $a_k^\sharp = e_{a_1(k)}^\sharp \fatsemi h_k^\sharp$ for all $k \in r(1)$. It follows that there exists h for which $a = h \fatsemi e$.

Solution to Exercise 4.39.

Here $p \in \textbf{Poly}$.

1. The canonical morphism $\eta \colon p \to p(1)$ is the identity $\eta_1 \colon p(1) \to p(1)$ on positions and the empty function on directions.
2. On positions, we have that $p_i(1)$ along with f_1 and g_1 form the following pullback square in **Set**:

$$\begin{tikzcd}
p_i(1) \arrow[r, "g_1"] \arrow[d, "f_1"'] \arrow[dr, phantom, "\lrcorner", very near start] & p(1) \arrow[d, equal] \\
1 \arrow[r, "i"'] & p(1)
\end{tikzcd}$$

So $p_i(1) \coloneqq \{(a, i') \in 1 \times p(1) \mid i = i'\} = \{(1, i)\}$, with f_1 uniquely determined and g_1 picking out $i \in p(1)$. Then on directions, we have that $p_i[(1,i)]$ along with $f^\sharp_{(1,i)}$ and $g^\sharp_{(1,i)}$ form the following pushout square in **Set**:

$$\begin{tikzcd}
p_i[(1,i)] \arrow[dr, phantom, "\lrcorner", very near start] & p[i] \arrow[l, "g^\sharp_{(1,i)}"'] \\
0 \arrow[u, "f^\sharp_{(1,i)}"] & 0 \arrow[l, "!"] \arrow[u, "!"']
\end{tikzcd}$$

So $p_i[(1,i)] \coloneqq p[i]$, with $f^\sharp_{(1,i)}$ uniquely determined and $g^\sharp_{(1,i)}$ as the identity. It follows that $p_i \coloneqq \{(1,i)\}y^{p[i]} \cong y^{p[i]}$, where $f \colon p_i \to 1$ is uniquely determined and $g \colon p_i \to p$ picks out $i \in p(1)$ on positions and is the identity on $p[i]$ on directions.

Solution to Exercise 4.40.

1. There are many possible answers, but one morphism $f \colon q \to r$, on positions, sends $1 \in q(1)$ (corresponding to y^2) to $2 \in r(1)$ (corresponding to 1) and $2 \in q(1)$ (corresponding to y) to $1 \in r(1)$ (corresponding to y). Then the on-directions functions $f_1^\sharp \colon 0 \to 2$ and $f_2^\sharp \colon 1 \to 1$ are uniquely determined. Another morphism $f' \colon q' \to r$, on positions, sends $1 \in q'(1)$ (corresponding to one of the y^3 terms) to $2 \in r(1)$ and both $2 \in q'(1)$ (corresponding to the other y^3 term) and $3 \in q'(1)$ (corresponding to the y^2 term) to $1 \in r(1)$. Then the on-directions function $(f')_1^\sharp \colon 0 \to 3$ is uniquely determined, while we can let $(f')_2^\sharp \colon 1 \to 3$ pick out 3 and $(f')_3^\sharp \colon 1 \to 2$ pick out 1.
2. We compute the pullback p along with the morphisms $g \colon p \to q$ and $g' \colon p \to q'$ of $q \xrightarrow{f} r \xleftarrow{f'} q'$ by following Example 4.37. We can compute $p(1)$ by taking the pullback in **Set**:

$$p(1) \coloneqq \{(i, i') \in 2 \times 3 \mid f_1(i) = f_1'(i)\} = \{(1,1), (2,2), (2,3)\}.$$

Moreover, the on-positions functions g_1 and g_1' send each pair in $p(1)$ to its left component and its right component, respectively.

To compute the direction-set at each p-position, we must compute a pushout. At $(1,1)$, we have $r[f_1(1)] = r[f_1'(1)] = r[2] = 0$, so the pushout $p[(1,1)]$ is just the sum $q[1] + q'[1] = 2 + 3 \cong 5$. Moreover, the on-directions functions $g^\sharp_{(1,1)}$ and $(g')^\sharp_{(1,1)}$ are the canonical inclusions $2 \to 2 + 3$ and $3 \to 2 + 3$.

At $(2,2)$, we have $r[f_1(2)] = r[f_1'(2)] = r[1] = 1$, with $f_2^\sharp$ picking out $1 \in 1 = q[2]$ and $(f')_2^\sharp$ picking out $3 \in 3 = q'[2]$. So the pushout $p[(2,2)]$ is the set $1 + 3 = \{(1,1), (2,1), (2,2), (2,3)\}$ but with $(1,1)$ identified with $(2,3)$; we can think of it as the set of equivalence classes $p[(2,2)] \cong \{\{(1,1), (2,3)\}, \{(2,1)\}, \{(2,2)\}\} \cong 3$. Moreover, the on-directions function $g^\sharp_{(2,2)}$ maps $1 \mapsto \{(1,1), (2,3)\}$, while the on-directions function $(g')^\sharp_{(2,2)}$ maps $1 \mapsto \{(2,1)\}, 2 \mapsto \{(2,2)\}$, and $3 \mapsto \{(1,1), (2,3)\}$.

Finally, at $(2,3)$, we have $r[f_1(2)] = r[f_1'(3)] = r[1] = 1$, with $f_2^\sharp$ still picking out $1 \in 1 = q[2]$ and $(f')_3^\sharp$ picking out $1 \in 2 = q'[3]$. So the pushout $p[(2,3)]$ is the set $1 + 2 = \{(1,1), (2,1), (2,2)\}$ but with $(1,1)$ identified with $(2,1)$; we can think of it as the set of equivalence classes $p[(2,3)] \cong$

$\{\{(1,1),(2,1)\},\{(2,2)\}\} \cong 2$. Moreover, the on-directions function $g^\sharp_{(2,3)}$ maps $1 \mapsto \{(1,1),(2,1)\}$, while the on-directions function $(g')^\sharp_{(2,3)}$ maps $1 \mapsto \{(1,1),(2,1)\}$ and $2 \mapsto \{(2,2)\}$.

It follows that $p \cong y^5 + y^3 + y^2$, with g and g' as described.

Solution to Exercise 4.41.

1. By Proposition 2.70, f_X (resp. g_X) sends each $(i,h) \in p(X)$ with $i \in p(1)$ and $h\colon p[i] \to X$ to $(f_1(i), f^\sharp_i \fatsemi h)$ (resp. $(g_1(i), g^\sharp_i \fatsemi h)$) in $q(X)$. So the equalizer of f_X and g_X is the set of all $(i,h) \in p(X)$ for which both $f_1(i) = g_1(i)$ and $f^\sharp_i \fatsemi h = g^\sharp_i \fatsemi h$.
   Indeed, by our construction of p', the set $p'(X)$ consists of all pairs (i,h') with $i \in p(1)$ such that $f_1(i) = g_1(i)$ and $h'\colon p'[i] \to X$, where $p'[i]$ is the coequalizer of $f^\sharp_i, g^\sharp_i\colon q[f_1(i)] \rightrightarrows p[i]$. By the universal property of the coequalizer, functions $h'\colon p'[i] \to X$ precisely correspond to functions $h\colon p[i] \to X$ for which $f^\sharp_i \fatsemi h = g^\sharp_i \fatsemi h$. So $p'(X)$ is indeed the equalizer of f_X and g_X.
   The equalizer natural transformation $e'\colon p' \to p$ has the inclusion $e'_X\colon p'(X) \to p(X)$ as its X-component, so by Corollary 2.72, it is the lens whose on-positions function is the canonical equalizer inclusion $e'_1\colon p'(1) \to p(1)$, while its on-directions function at $i \in p'(1)$ is the map $p[i] \to p'[i]$ corresponding to the identity on $p'[i]$ given by the universal property of the coequalizer—which is just the canonical coequalizer map $p[i] \to p'[i]$. But this is exactly the map $e\colon p' \to p$ constructed in the proof of Proposition 2.79, as desired.
2. By Proposition 2.33, limits—including equalizers—in $[\mathbf{Set},\mathbf{Set}]$ are computed pointwise. So if $e_X\colon p'(X) \to p(X)$ is the equalizer of $f_X, g_X\colon p(X) \rightrightarrows q(X)$ for every $X \in \mathbf{Set}$, then $e\colon p' \to p$ is the equalizer of $f, g\colon p(X) \rightrightarrows q(X)$.
3. We have just shown that equalizers in **Poly** coincide with equalizers in [**Set**, **Set**]. We saw in the proof of Proposition 2.79 that products in **Poly** also coincide with products in [**Set**, **Set**]. Since every limit can be computed as an equalizer of products, we can conclude that limits in **Poly** coincide with limits in [**Set**, **Set**].

Solution to Exercise 4.43.

1. We define a map $g\colon q \to q'$ as follows. The on-positions function $g_1\colon q(1) \to q'(1)$ is the coequalizer of $s_1, t_1\colon p(1) \rightrightarrows q(1)$. In particular, g_1 sends each vertex in $q(1)$ to its corresponding connected component in $q'(1) = C$. Then for each $v \in q(1)$, if we let its corresponding connected component be $c \coloneqq g_1(v)$, we can define the on-directions function $g^\sharp_v\colon q'[c] \to q[v]$ to be the projection from the limit $q'[c]$ to its component $q[v]$.
2. To show that $s \fatsemi g = t \fatsemi g$, we must show that both sides are equal on positions and on directions. The on-positions function g_1 is defined to be the coequalizer of s_1 and t_1, so $s_1 \fatsemi g_1 = t_1 \fatsemi g_1$. So it suffices to show that for all $e \in p(1)$, if we let its corresponding connected component be $c \coloneqq g_1(s_1(e)) = g_1(t_1(e))$, then the following diagram of on-directions functions commutes:
$$\begin{tikzcd}
 & q[s_1(e)] \arrow[dl, "s^\sharp_e"'] & \\
p[e] & & q'[c] \arrow[ul, "g^\sharp_{s_1(e)}"'] \arrow[dl, "g^\sharp_{t_1(e)}"] \\
 & q[t_1(e)] \arrow[ul, "t^\sharp_e"] &
\end{tikzcd}$$
   But this is automatically true by the definition of $q'[c]$ as a limit—specifically the limit of a functor with $s^\sharp_e$ and $t^\sharp_e$ in its image—and the definitions of $g^\sharp_{s_1(e)}$ and $g^\sharp_{t_1(e)}$ as projections from this limit.
3. To show that g is the coequalizer of s and t, it suffices to show that for any $r \in \mathbf{Poly}$ and map $f\colon q \to r$ satisfying $s \fatsemi f = t \fatsemi f$, there exists a unique map $h\colon q' \to r$ for which $f = g \fatsemi h$, so that the following diagram commutes.
$$\begin{tikzcd}
p \arrow[r, shift left, "s"] \arrow[r, shift right, "t"'] & q \arrow[r, "g"] \arrow[dr, "f"'] & q' \arrow[d, dashed, "h"] \\
 & & r
\end{tikzcd}$$

In order for $f = g \fatsemi h$ to hold, we must have $f_1 = g_1 \fatsemi h_1$ on positions. But we have that $s_1 \fatsemi f_1 = t_1 \fatsemi f_1$, so by the universal property of $q'(1)$ and the map g_1 as the coequalizer of s_1 and t_1 in **Set**, there exists a unique h_1 for which $f_1 = g_1 \fatsemi h_1$. Hence h is uniquely characterized on positions. In particular, it must send each connected component $c \in q'(1)$ to the element in $r(1)$ to which f_1 sends every vertex $v \in V_c = g_1^{-1}(c)$ that lies in the connected component c.

Then for $f = g \fatsemi h$ to hold on directions, we must have that $f_v^\sharp = h^\sharp_{g_1(v)} \fatsemi g_v^\sharp$ for each $v \in q(1)$. Put another way, given $c \in q'(1)$, we must have that $f_v^\sharp = h_c^\sharp \fatsemi g_v^\sharp$ for every $v \in V_c$. But $s \fatsemi f = t \fatsemi f$ implies that for each $e \in E_c = s_1^{-1}(g_1^{-1}(c)) = t_1^{-1}(g_1^{-1}(c)) \subseteq p(1)$, the following diagram of on-directions functions commutes:

$$\begin{tikzcd}
& q[s_1(e)] \arrow[dl, "s_e^\sharp"'] & \\
p[e] & & r[f_1(v)] \arrow[ul, "f^\sharp_{s_1(e)}"'] \arrow[dl, "f^\sharp_{t_1(e)}"] \\
& q[t_1(e)] \arrow[ul, "t_e^\sharp"] &
\end{tikzcd}$$

It follows that $r[f_1(v)]$ together with the maps $(f_v^\sharp)_{v \in V_c}$ form a cone over the functor F. So by the universal property of the limit $q'[c]$ of F with projection maps $(g_v^\sharp)_{v \in V_c}$, there exists a unique $h_c^\sharp \colon r[f_1(v)] \to q'[c]$ for which $f_v^\sharp = h_c^\sharp \fatsemi g_v^\sharp$ for every $v \in V_c$. Hence h is also uniquely characterized on directions, so it is unique overall. Moreover, we have shown that we can define h on positions so that $f_1 = g_1 \fatsemi h_1$, and that we can define h on directions such that $f_v^\sharp = h_c^\sharp \fatsemi g_v^\sharp$ for all $c \in q'(1)$ and $v \in V_c$. It follows that there exists h for which $f = g \fatsemi h$.

Solution to Exercise 4.46.

1. We characterize the map $\epsilon \colon p(1)y \to p$ as follows. On positions, it is the identity on $p(1)$. Then for each $i \in p(1)$, on directions, it is the unique map $p[i] \to 1$.
2. We characterize the map $\eta \colon p \to y^{\Gamma(p)}$ as follows. On positions, it is the unique map $p(1) \to 1$. Then for each $i \in p(1)$, on directions, it is the canonical projection $\Gamma(p) \cong \prod_{i' \in p(1)} p[i'] \to p[i]$.
3. Showing that (4.47) is a pushout square is equivalent to showing that, in the diagram

$$\begin{tikzcd}
& y \arrow[d, "\iota"] \arrow[dr, "!"] & \\
p(1)y \arrow[ur, "!"] \arrow[r, shift left, "s"] \arrow[r, shift right, "t"'] \arrow[dr, "\epsilon"'] & y + p \arrow[r, "g"] & y^{\Gamma(p)} \\
& p \arrow[u, "\iota'"'] \arrow[ur, "\eta"'] &
\end{tikzcd} \tag{4.74}$$

in which ι, ι' are the canonical inclusions and the four triangles commute, $y^{\Gamma(p)}$ equipped with the morphism g is the coequalizer of s and t. To do so, we apply Theorem 4.42 to compute the coequalizer q' of s and t. The position-set of q' is the coequalizer of $s_1 = (! \fatsemi \iota)_1$, which sends every $i \in p(1)$ to the position of $y + p$ corresponding to the summand y, and $t_1 = (\epsilon \fatsemi \iota')_1$, which sends each $i \in p(1)$ to the corresponding position in the summand p of $y + p$. It follows that the coequalizer of s_1 and t_1 is 1, so $q'(1) \cong 1$.

Then the direction-set of q' at its sole position is the limit of the functor F whose image consists of morphisms of the form $1 \to 1$ or $p[i] \to 1$ for every $i \in p(1)$. It follows that the limit of F is just a product, namely $\prod_{i \in p(1)} p[i] \cong \Gamma(p)$. Hence $q' \cong y^{\Gamma(p)}$, as desired.

It remains to check that the upper right and lower right triangles in (4.74) commute. The upper right triangle must commute by the uniqueness of morphisms $y \to y^{\Gamma(p)}$; and the lower right triangle must commute on positions. Moreover, the on-directions function of the coequalizer morphism g at each position $i \in p(1) \subseteq (y + p)(1)$ must be the canonical projection $\Gamma(p) \to p[i]$, which matches the behavior of the corresponding on-directions function of η; hence the lower right triangle also commutes on directions.

Solution to Exercise 4.53.

Consider the maps $y \xrightarrow{f} y^2 + y \xrightarrow{g} y$ where f is the canonical inclusion and g is uniquely determined on positions and picks out $1 \in 2$ and $1 \in 1$ on directions. Then the only on-directions function of f is a function $1 \to 1$, an isomorphism, so f is cartesian. Meanwhile, one of the on-directions functions of g is a function $1 \to 2$, which is not an isomorphism, so g is not cartesian. Finally, $f \mathbin{⨾} g$ can only be the unique morphism $y \to y$, namely the identity, which is cartesian.

Solution to Exercise 4.55.

We wish to show that a morphism $f\colon p \to q$ in **Poly** is cartesian if and only if the square on the left hand side of (4.54) is a pullback. We already know that that square commutes, so it is a pullback if and only if $f^\sharp$ is an isomorphism. The right pullback square tells us that the $\bullet$ is $\sum_{i \in p(1)} q[f_1(i)]$. So $f_i^\sharp\colon q[f_1(i)] \to p[i]$ is an isomorphism for every $i \in p(1)$ if and only if their sum $f^\sharp\colon \sum_{i \in p(1)} q[f_1(i)] \to \sum_{i \in p(1)} p[i] \cong \dot{p}(1)$ is an isomorphism as well. Hence f is cartesian if and only if $f^\sharp$ is an isomorphism, as desired.

Solution to Exercise 4.56.

The pushout of a cartesian map is *not* necessarily cartesian. Take the pushout square (4.47). The map $!\colon p(1)y \to y$ has $1 \to 1$ as every on-directions function, so it is cartesian, but its pushout $\eta\colon p \to y^{\Gamma(p)}$ is not going to be cartesian as long as there is some $i \in p(1)$ for which $\Gamma(p) \not\cong p[i]$. For instance, when $p \coloneqq y + 1$, we have that $\Gamma(p) \cong 0 \not\cong 1 \cong p[1]$, so η is not cartesian.

Solution to Exercise 4.60.

First, we will show that $1 \Rightarrow 3$ in Proposition 4.58. Here $f\colon p \to q$ is a cartesian morphism in **Poly** and $h\colon A \to B$ is a function.

1. An element of $p(B)$ is a pair comprised of a p-position i and a function $k\colon p[i] \to B$, and Proposition 2.70 tells us that $f_B\colon p(B) \to q(B)$ sends $(i, k) \mapsto (f_1(i), f_i^\sharp \mathbin{⨾} k)$. Meanwhile, an element of $q(A)$ is a pair comprised of a q-position j and a function $\ell\colon q[j] \to A$, and Proposition 2.46 tells us that $q(h)$ sends $(j, \ell) \mapsto (j, \ell \mathbin{⨾} h)$. So $((i, k), (j, \ell))$ is in the pullback of $p(B) \xrightarrow{f_B} q(B) \xleftarrow{q(h)} q(A)$ if and only if $f_1(i) = j$ and $f_i^\sharp \mathbin{⨾} k = \ell \mathbin{⨾} h$.

   As f is cartesian, $f_i^\sharp$ is an isomorphism, so we can rewrite the latter equation as $k = g_i \mathbin{⨾} \ell \mathbin{⨾} h$, where g_i is the inverse of $f_i^\sharp$. In fact, if we let $\ell' \coloneqq g_i \mathbin{⨾} \ell$, we observe that the values of j, k, and ℓ are all already determined by the values of i and ℓ': we have that $j = f_1(i)$, that $k = \ell' \mathbin{⨾} h$, and that $\ell = f_i^\sharp \mathbin{⨾} \ell'$ It follows that the pullback is equivalently the set of pairs (i, ℓ') comprised of a p-position i and a function $\ell'\colon p[i] \to A$ (with no other restrictions on i and ℓ'). The projection from the pullback to $p(B)$ sends $(i, \ell') \mapsto (i, \ell' \mathbin{⨾} h)$, and the projection from the pullback to $q(A)$ sends $(i, \ell') \mapsto (f_1(i), f_i^\sharp \mathbin{⨾} \ell')$.
2. The pullback described above—the set of pairs (i, ℓ') comprised of a p-position i and a function $\ell'\colon p[i] \to A$—is exactly the set $p(A)$. Moreover, the projection to $p(B)$ sending $(i, \ell') \mapsto (i, \ell' \mathbin{⨾} h)$ is $p(h)$, and the projection to $q(A)$ sending $(i, \ell') \mapsto (f_1(i), f_i^\sharp \mathbin{⨾} \ell')$ is f_A by Proposition 2.70. So (4.59) is a pullback, as desired.

Next, we will show that $3 \Rightarrow 1$ in Proposition 4.58, with $f\colon p \to q$ as a morphism in **Poly** that is a cartesian natural transformation and $i \in p(1)$.

3. By Corollary 2.72, $f_{p[i]}$ sends $(i, \mathrm{id}_{p[i]}) \mapsto (f_1(i), f_i^\sharp)$, and by Proposition 2.46, $q(f_i^\sharp)$ sends $(f_1(i), \mathrm{id}_{q[f_1(i)]}) \mapsto (f_1(i), f_i^\sharp)$ as well, so (4.61) commutes.
4. Taking $A \coloneqq q[f_1(i)], B \coloneqq p[i]$, and $h \coloneqq f_i^\sharp$ in (4.59) and applying its universal property to (4.61) induces an element (i', g) of $p(q[f_1(i)])$, with $i' \in p(1)$ and $g\colon p[i'] \to q[f_1(i)]$, such that $p(f_i^\sharp)$ sends $(i', g) \mapsto (i, \mathrm{id}_{p[i]})$ and $f_{q[f_1(i)]}$ sends $(i', g) \mapsto (f_1(i), \mathrm{id}_{q[f_1(i)]})$. It follows from the behavior of $p(f_i^\sharp)$ (by Proposition 2.46) that $i' = i$ and $g \mathbin{⨾} f_i^\sharp = \mathrm{id}_{p[i]}$, and it follows from the behavior of

$f_{q[f_1(i)]}$ (by Proposition 2.70) that $f_i^\sharp \mathbin{⨟} g = \mathrm{id}_{q[f_1(i)]}$. So g is the inverse of $f_i^\sharp$, proving that $f_i^\sharp$ is an isomorphism, as desired.

Solution to Exercise 4.69.

Choose $p \in \mathbf{Poly}$ and $q' \in \mathbf{Poly}/y$. Then there is $q \in \mathbf{Poly}$ such that $q' \cong qy$, equipped with the projection $qy \to y$. The pushforward is given by the exponential

$$f_*(qy) \coloneqq q^y$$

from the cartesian closure; see (4.27). Indeed, we have

$$\begin{aligned}\mathbf{Poly}/y(f^*p, qy) &\cong \mathbf{Poly}/y(py, qy)\\ &\cong \mathbf{Poly}(py, q)\\ &\cong \mathbf{Poly}(p, q^y).\end{aligned}$$

Part II

A different category of categories

Chapter 5

The composition product

We have seen that the category **Poly** of polynomial functors has quite a bit of well-interoperating mathematical structure. Further, it is an expressive way to talk about dynamical systems that can change their interfaces and wiring patterns based on their internal states.

But we touched upon one thing—what in some sense is the most interesting part of the story—only briefly. That thing is quite simple to state, and yet has profound consequences. Namely, polynomials can be composed:

$$y^2 \circ (y+1) = (y+1)^2 \cong y^2 + 2y + 1.$$

What could be simpler?

It turns out that this operation, which we'll see soon is a monoidal product, has a lot to do with time. There is a strong sense—made precise in Proposition 5.2—in which the polynomial $p \circ q$ represents "starting at a position i in p, choosing a direction in $p[i]$, landing at a position j in q, choosing a direction in $q[j]$, and then landing... somewhere."

The composition product has many surprises up its sleeve, as we'll see in the following chapters. We've given you a glimpse of many of them already in Section 1.6. We won't amass them all here; instead, we'll take you through the story step by step. But as a preview, $\circ$ will get us into decision trees, databases, and more dynamics, as well as the interactions between these.

5.1 Defining the composition product

We begin with the definition of the composition product in terms of polynomials as functors.

5.1.1 Composite functors

Definition 5.1. Given polynomial functors p, q, we let $p \circ q$ denote their composite as functors. That is, $p \circ q \colon \mathbf{Set} \to \mathbf{Set}$ sends each set X to the set $p(q(X))$.

Functor composition gives a monoidal structure on the category $\mathbf{Set}^{\mathbf{Set}}$ of functors $\mathbf{Set} \to \mathbf{Set}$, but to check that the full subcategory **Poly** of $\mathbf{Set}^{\mathbf{Set}}$ inherits this monoidal structure, we need to verify that the composite of two functors in **Poly** is still a functor in **Poly**.

Proposition 5.2. Suppose $p, q \in \mathbf{Poly}$ are polynomial functors $p, q \colon \mathbf{Set} \to \mathbf{Set}$. Then their composite $p \circ q$ is again a polynomial functor, and we have the following isomorphism:

$$p \circ q \cong \sum_{i \in p(1)} \prod_{d \in p[i]} \sum_{j \in q(1)} \prod_{e \in q[j]} y. \tag{5.3}$$

Proof. We can rewrite p and q as

$$p \cong \sum_{i \in p(1)} y^{p[i]} \cong \sum_{i \in p(1)} \prod_{d \in p[i]} y \qquad \text{and} \qquad q \cong \sum_{j \in q(1)} y^{q[j]} \cong \sum_{j \in q(1)} \prod_{e \in q[j]} y.$$

For any set X we have $(p \circ q)(X) = p(q(X)) = p(\sum_j \prod_e X) = \sum_i \prod_d \sum_j \prod_e X$, so (5.3) is indeed the formula for the composite $p \circ q$. To see this is a polynomial, we use (2.26), which says we can rewrite the $\prod \sum$ in (5.3) as a $\sum \prod$ to obtain

$$p \circ q \cong \sum_{i \in p(1)} \sum_{\bar{j}_i \colon p[i] \to q(1)} y^{\sum_{d \in p[i]} q[\bar{j}_i(d)]} \tag{5.4}$$

(written slightly bigger for clarity), which is clearly a polynomial. □

Corollary 5.5. The category **Poly** has a monoidal structure $(y, \circ)$, where y is the identity functor and $\circ$ is given by composition.

Because we may wish to use $\circ$ to denote composition in arbitrary categories, we use a special symbol for polynomial composition, namely

$$p \triangleleft q \coloneqq p \circ q.$$

The symbol $\triangleleft$ looks a bit like the composition symbol, in that it is an open shape, and when writing quickly by hand, it's okay if it morphs into a $\circ$. But $\triangleleft$ highlights the asymmetry of composition, in contrast with the other monoidal structures on **Poly** we've encountered. Moreover, we'll soon see that $\triangleleft$ is quite evocative in terms of trees. For each $n \in \mathbb{N}$, we'll also use $p^{\triangleleft n}$ to denote the n-fold composite of p, i.e. n copies of p all composed with each other. In particular, $p^{\triangleleft 0} \coloneqq y$ and $p^{\triangleleft 1} \coloneqq p$.

We repeat the important formulas from Proposition 5.2 and its proof in the new notation:

$$p \triangleleft q \cong \sum_{i \in p(1)} \prod_{d \in p[i]} \sum_{j \in q(1)} \prod_{e \in q[j]} y. \tag{5.6}$$

$$p \triangleleft q \cong \sum_{i \in p(1)} \sum_{\bar{j}_i \colon p[i] \to q(1)} y^{\sum_{d \in p[i]} q[\bar{j}_i(d)]} \tag{5.7}$$

Exercise 5.8 (Solution here). Let's consider (5.7) piece by piece, with concrete polynomials $p := y^2 + y^1$ and $q := y^3 + 1$.

1. What is $y^2 \triangleleft q$?
2. What is $y^1 \triangleleft q$?
3. What is $(y^2 + y^1) \triangleleft q$? This is what $p \triangleleft q$ "should be."
4. How many functions $\bar{j}_1 \colon p[1] \to q(1)$ are there?
5. For each function $\bar{j}_1$ as above, what is $\sum_{d \in p[1]} q[\bar{j}_1(d)]$?
6. How many functions $\bar{j}_2 \colon p[2] \to q(1)$ are there?
7. For each function $\bar{j}_2$ as above, what is $\sum_{d \in p[2]} q[\bar{j}_2(d)]$?
8. Write out
$$\sum_{i \in p(1)} \sum_{\bar{j}_i \colon p[i] \to q(1)} y^{\sum_{d \in p[i]} q[\bar{j}_i(d)]}.$$
Does the result agree with what $p \triangleleft q$ should be? ◊

Exercise 5.9 (Solution here).

1. If p and q are representable, show that $p \triangleleft q$ is too. Give a formula for it.
2. If p and q are linear, show that $p \triangleleft q$ is too. Give a formula for it.
3. If p and q are constant, show that $p \triangleleft q$ is too. Give a formula for it. ◊

We have how $\triangleleft$ acts on the objects in **Poly**, but what does it do to the morphisms between them? For any pair of natural transformations $f \colon p \to p'$ and $g \colon q \to q'$ between polynomial functors, their composition product $f \triangleleft g \colon p \triangleleft q \to p' \triangleleft q'$ is given by *horizontal composition*.

Definition 5.10 (Horizontal composition of natural transformations)**.** Let $f \colon p \to p'$ and $g \colon q \to q'$ be two natural transformations between (polynomial) functors $p, p', q, q' \colon \mathbf{Set} \to \mathbf{Set}$. Then the *horizontal composite* of f and g, denoted $f \triangleleft g$, is the natural transformation $p \triangleleft q \to p' \triangleleft q'$ whose X-component for each $X \in \mathbf{Set}$ is the function

$$p(q(X)) \xrightarrow{f_{q(X)}} p'(q(X)) \xrightarrow{p'(g_X)} p'(q'(X)) \tag{5.11}$$

obtained by composing the $q(X)$-component of f with p' applied to the X-component of g.

Exercise 5.12 (Solution here). Show that we could have replaced the composite function (5.11) in Definition 5.10 with the function

$$p(q(X)) \xrightarrow{p(g_X)} p(q'(X)) \xrightarrow{f_{q'(X)}} p'(q'(X)) \tag{5.13}$$

obtained by composing p applied to the X-component of g with the $q'(X)$-component of f, without altering the definition. ◊

The composition product of polynomials and lenses will be extremely important in the story that follows. However, we only sometimes think of it as the composition of functors and the horizontal composition of natural transformations; more often we think of it as certain operations on arenas or corolla forests.

5.1.2 Composite arenas

Let us interpret our formula (5.7) for the composite of two polynomials in terms of what it says about the positions and directions of the corresponding arenas. The position-set of $p \triangleleft q$ is

$$(p \triangleleft q)(1) \cong \sum_{i \in p(1)} \sum_{\bar{j}_i \colon p[i] \to q(1)} 1 \cong \sum_{i \in p(1)} \mathbf{Set}(p[i], q(1)). \tag{5.14}$$

In other words, specifying a position of $p \triangleleft q$ amounts to first specifying a p-position i, then specifying a function $\bar{j}_i \colon p[i] \to q(1)$, i.e. a q-position $\bar{j}_i(d)$ for each $p[i]$-direction d.

Given such a position $(i, \bar{j}_i)$ of $p \triangleleft q$, the direction-set of $p \triangleleft q$ at $(i, \bar{j}_i)$ is

$$(p \triangleleft q)[(i, \bar{j}_i)] \cong \sum_{d \in p[i]} q[\bar{j}_i(d)]. \tag{5.15}$$

So a direction of $p \triangleleft q$ at $(i, \bar{j}_i)$ consists of a $p[i]$-direction d and a $q[\bar{j}_i(d)]$-direction.

While this description completely characterizes $p \triangleleft q$ as an arena, it may be a bit tricky to wrap your head around. Here is an alternative perspective that can help us get a better intuition for what's going on with composite arenas.

Back in Section 2.2.1, we saw how to write the instructions for choosing an element of a dependent sum or product of sets. For instance, given a polynomial p and a set X, the instructions for choosing an element of

$$p \triangleleft X = p(X) \cong \sum_{i \in p(1)} \prod_{d \in p[i]} X$$

would be written as follows.

To choose an element of $p(X)$:

1. choose an element $i \in p(1)$;
2. for each element $d \in p[i]$:
    2.1. choose an element of X.

But say we hadn't picked a set X yet; in fact, say we might replace X with a general polynomial instead. We'll replace "an element of X" with a placeholder—the words "a future"—that indicates that we don't yet know what will go there. Furthermore, to highlight that these instructions are associated with some polynomial p, we will use our familiar arena terminology of positions and directions.

> The instructions associated with a polynomial p are:
>
> 1. choose a p-position i;
> 2. for each $p[i]$-direction d:
>   2.1. choose a future.

If we think of polynomials in terms of their instructions, then (5.6) tells us that the composition product simply nests one set of instructions within another, as follows.

> The instructions associated with a polynomial $p \triangleleft q$ are:
>
> 1. choose a p-position i;
> 2. for each $p[i]$-direction d:
>   2.1. choose a q-position j;
>   2.2. for each $q[j]$-direction e:
>   2.2.1. choose a future.

Similarly, we could write down the instructions associated with any n-fold composite by nesting even further. We might think of such instructions as specifying some sort of length-n *strategy*, in the sense of game theory, for picking positions given any directions—except that the opponent is somehow abstract, having no positions of its own.

When we rewrite (5.6) (5.7), we are collapsing the instructions down into the following, highlighting the positions and directions of $p \triangleleft q$.

> The instructions associated with a polynomial $p \triangleleft q$ are:
>
> 1. choose a p-position i and, for each $p[i]$-direction d, a q-position $\bar{j}_i(d)$;
> 2. for each $p[i]$-direction d and each $q[\bar{j}_i(d)]$-direction e:
>   2.1. choose a future.

We will see in Section 5.1.3 that these instructions have a very natural interpretation when we translate from arenas to corolla forests.

Exercise 5.16 (Solution here).

1. Let p be an arbitrary polynomial. Write out the (uncollapsed) instructions associated with $p^{\triangleleft 3} = p \triangleleft p \triangleleft p$.

2. Write out the (uncollapsed) instructions for choosing an element of $p \triangleleft p \triangleleft 1$, but where you would normally write "choose an element of 1," just write "done." $\Diamond$

But how does the composition product act on lenses between arenas? Given lenses $f\colon p \to p'$ and $g\colon q \to q'$, we can translate them to natural transformations, take their horizontal composite, then translate this back to a lens. The following exercise guides us through this process.

Exercise 5.17 (The composition product of lenses; solution here). Fix lenses $f\colon p \to p'$ and $g\colon q \to q'$. We seek to characterize their composition product $f \triangleleft g\colon p \triangleleft q \to p' \triangleleft q'$.

1. Use Proposition 2.70 to compute the $q(X)$-component of f as a natural transformation.
2. Use Propositions 2.46 and 2.70 to compute p' applied to the X-component of g as a natural transformation.
3. Combine #1 and #2 using Definition 5.10 to compute the horizontal composite $f \triangleleft g$ of f and g as natural transformations.
4. Use Corollary 2.72 to translate the natural transformation $f \triangleleft g$ obtained in #3 to a lens between arenas $p \triangleleft q \to p' \triangleleft q'$. Verify that for each $(i, \bar{j}_i)$ in $(p \triangleleft q)(1)$ (see (5.14)), its on-positions function sends

$$(i, \bar{j}_i) \xmapsto{(f \triangleleft g)_1} (f_1(i), f_i^\sharp \fatsemi \bar{j}_i \fatsemi g_1); \tag{5.18}$$

while for each (d', e') in $(p' \triangleleft q')[(f_1(i), f_i^\sharp \fatsemi \bar{j}_i \fatsemi g_1)]$ (see (5.15)), its on-directions function sends

$$(d', e') \xmapsto{(f \triangleleft g)^\sharp_{(i,\bar{j}_i)}} \left(f_i^\sharp(d'), g^\sharp_{\bar{j}_i(f_i^\sharp(d'))}(e')\right). \tag{5.19}$$

$\Diamond$

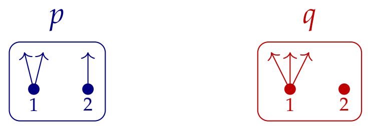
So what does Exercise 5.17 tell us about the behavior of $f \triangleleft g\colon p \triangleleft q \to p' \triangleleft q'$? By (5.18), on positions, $f \triangleleft g$ takes a p-position i and sends it to the p'-position $f_1(i)$; then for each direction d' at this position, the associated q'-position is obtained by sending d' back to a $p[i]$-direction via $f_i^\sharp$, checking what q-position is associated to that $p[i]$-direction via some $\bar{j}_i$, then sending that q-position forward again to a q'-position via g_1.

Then by (5.18), on directions, $f \triangleleft g$ sends a direction of p' back to a direction of p via an on-directions function of f, then sends a direction of q' back to a direction of q via an on-directions funtion of g. We'll get a better sense of what's happening when we see this drawn out as corolla forests in Example 5.28.

5.1.3 Composite corolla forests

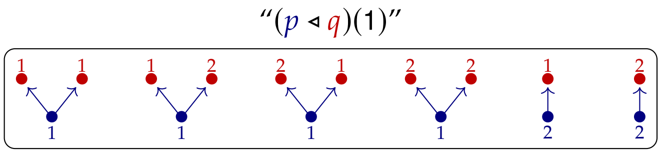
It turns out that the forest of $p \triangleleft q$ is given by gluing corollas from the forest of q onto the leaves of corollas from the forest of p in every possible way. We will demonstrate this using an example.

Let's say $p \coloneqq y^2 + y$ and $q \coloneqq y^3 + 1$, whose corolla forests we draw as follows:



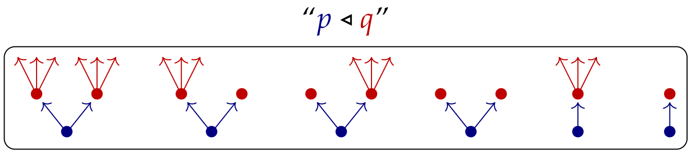
(5.20)

By (5.14), choosing a position of $p \triangleleft q$ amounts to first choosing a root i from the forest of p, then choosing a root from the forest of q for every leaf emanating from i. So we may depict $(p \triangleleft q)(1)$ by gluing roots from q to leaves in p in every possible way, as follows:



(5.21)

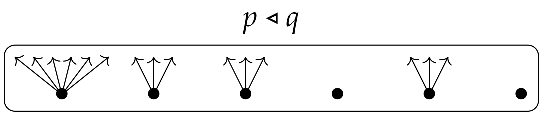
Now fix one of the positions of $p \triangleleft q$ drawn above: a root i from p and a root from q glued to every leaf emanating from i. By (5.15), a direction of $p \triangleleft q$ at that position consists of a leaf d emanating from the root i from p and a second leaf emanating from the root from q that has been glued to d. In other words, in the following picture, where we have glued not just roots but entire corollas from q to leaves in p, the directions of $p \triangleleft q$ at the position corresponding to each tree are the rooted paths[1] of that tree of length 2 (we omit the labels):



(5.22)

Equivalently, we can think of the directions in the picture above as the leaves at the second level of each tree. So $p \triangleleft q$ has six positions; the first has six directions, the second, third, and fifth have three directions, and the fourth and sixth have no directions. In total, we can read off that $p \triangleleft q$ is isomorphic to $y^6 + 3y^3 + 2$.

We put the $p \triangleleft q$ in scare quotes above (5.22) because, to be pedantic, the corolla forest of $p \triangleleft q$ has the two levels smashed together as follows:



(5.23)

Usually, we will prefer the style of (5.22) rather than the more pedantic style of (5.23).

We have now seen how to draw a single polynomial as a corolla forest, with level-1 leaves as directions; as well as how to draw a two-fold composite of polynomials as

[1] A *rooted path* of a rooted tree is a path up the tree that starts from the root.

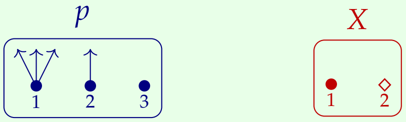
a forest of trees, with level-2 leaves as directions. Note that drawing a corolla of p or a tree of $p \triangleleft q$ is just a graphical way of following the instructions associated with the polynomial p or $p \triangleleft q$ that we saw in Section 5.1.2, where the arrows—the top-level leaves—are where the "futures" would go. Similarly, we could depict any n-fold composite as a forest with level-n leaves as directions. You'll have an opportunity to try this in the following exercise.

Exercise 5.24 (Solution here). Use p, q as in (5.20) and $r := 2y + 1$ in the following.

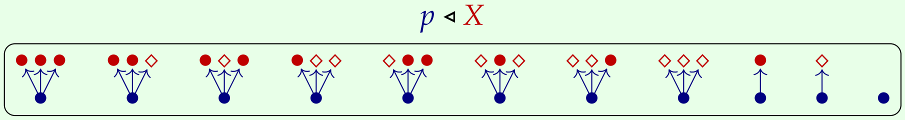
1. Draw $q \triangleleft p$.
2. Draw $p \triangleleft p$.
3. Draw $p \triangleleft p \triangleleft 1$.
4. Draw $r \triangleleft r$.
5. Draw $r \triangleleft r \triangleleft r$. ◊

Example 5.25 (Composing polynomials with constants). For any set X and polynomial p, we can take $p(X) \in \mathbf{Set}$; indeed $p\colon \mathbf{Set} \to \mathbf{Set}$ is a functor! In particular, by this point you've seen us write $p(1)$ hundreds of times. But we've also seen that X is itself a polynomial, namely a constant one.

It's not hard to see that $p(X) \cong p \triangleleft X$. Here's a picture, where $p := y^3 + y + 1$ and $X := 2$.

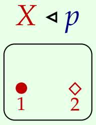


Let's see how $(y^3 + y + 1) \triangleleft 2$ looks.



It has 11 positions and no level-2 leaves, which means it's a set (constant polynomial, with no directions), namely $p \triangleleft X \cong 11$.

We could also draw $X \triangleleft p$, since both are perfectly valid polynomials. Here it is:



Each of the leaves in X—of which there are none—is given a corolla from p.

Exercise 5.26 (Solution here).

1. Choose a polynomial p and draw $p \triangleleft 1$ in the style of Example 5.25.
2. Show that if X is a set (considered as a constant polynomial) and p is any polynomial, then $X \triangleleft p \cong X$.
3. Show that if X is a set and p is a polynomial, then $p \triangleleft X \cong p(X)$, where $p(X)$ is the set given by applying p as a functor to X. ◊

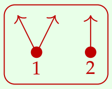
Exercise 5.27 (Solution here). For any $p \in \mathbf{Poly}$ there are natural isomorphisms $p \cong p \triangleleft y$ and $p \cong y \triangleleft p$.

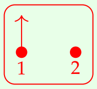
1. Thinking of polynomials as functors $\mathbf{Set} \to \mathbf{Set}$, what functor does y represent?
2. Why are $p \triangleleft y$ and $y \triangleleft p$ isomorphic to p?
3. Let $p := y^3 + y + 1$. In terms of tree pictures, draw $p \triangleleft y$ and $y \triangleleft p$, and explain pictorially how to see the isomorphisms $p \triangleleft y \cong p \cong y \triangleleft p$. ◊

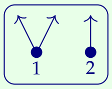
This is just p with every position propped up one level, so it is also still a picture of p.

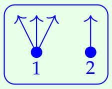
How shall we think about taking the composition product of lenses in terms of our tree pictures? We can interpret the results of Exercise 5.17 as follows.

Example 5.28. Let's take $p := y^2 + y$, $q := y^2 + y$, $p' := y^3 + y$, and $q' := y + 1$.

$q =$


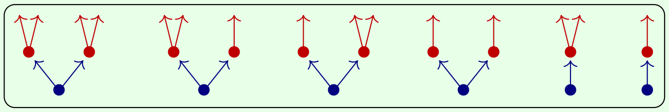
$q' =$


$p =$


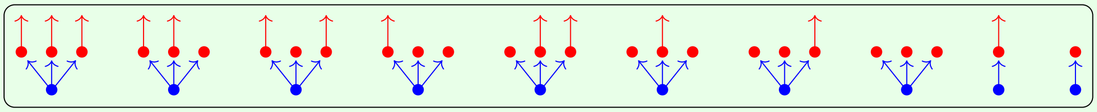
$p' =$


For any pair of lenses $p \to p'$ and $q \to q'$, we have a lens $p \triangleleft q \to p' \triangleleft q'$. Let's draw $p \triangleleft q$ and $p' \triangleleft q'$.

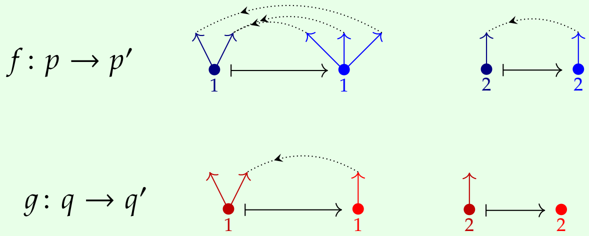
$p \triangleleft q$



$p' \triangleleft q'$



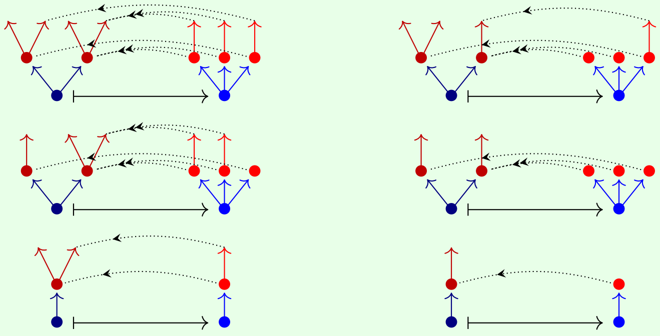
Let's also pick a pair of lenses, $f\colon p \to p'$ and $g\colon q \to q'$.



Then by Exercise 5.17, we can form the lens $f \triangleleft g\colon p \triangleleft q \to p' \triangleleft q'$ as follows. On positions, we follow (5.18): for each tree t in the picture of $p \triangleleft q$, we begin by using f_1 to send the corolla i of p that forms the bottom level of t to a corolla i' of p'. Then for each leaf d' of i', to choose what corolla of q' gets glued to d', we use $f_i^\sharp$ to send d' back to a leaf d of corolla i. Since t has corolla i as its bottom level, d is just a level-1 vertex of the tree t. So we can take the corolla j of q glued to d in t, then use g_1 to send j forward to a corolla j' of q'. This is the corolla we glue to d'. All this specifies a tree t' in $p' \triangleleft q'$ that t gets sent to via $(f \triangleleft g)_1$.

On directions, we follow (5.19): picking a direction of t' consists of picking a level-1 vertex d' and a level-2 leaf e' emanating from d'. The on-directions function $f_i^\sharp$ sends d' back to a level-1 vertex d of t, and as we saw, the on-positions function g_1 sends the corolla j of q glued to d in t forward to the corolla of q' glued to d'. Then e' is a leaf of that corolla, and $g_j^\sharp$ sends e' back to a leaf e emanating from d. So the on-directions function $(f \triangleleft g)_t^\sharp$ sends the level-2 leaf e' to the level-2 leaf e.

We draw the lens $f \triangleleft g \to p \triangleleft q \to p' \triangleleft q'$ below. To avoid clutter, we leave out the arrows for g_1 that show how the red corollas on the right are selected; we hope the reader can put it together for themselves.



Exercise 5.29 (Solution here). With p, q, p', q' and f, g as in Example 5.28, draw the lens $g \triangleleft f\colon q \triangleleft p \to q' \triangleleft p'$ in terms of trees as in the example. ◊

Exercise 5.30 (Solution here). Suppose p, q, and r are polynomials and you're given arbitrary lenses $f\colon q \to p \triangleleft q$ and $g\colon q \to q \triangleleft r$. Does the following diagram necessarily commute?[a]

$$\begin{tikzcd}
q \arrow[r, "g"] \arrow[d, "f"'] \arrow[dr, phantom, "?"] & q \triangleleft r \arrow[d, "f \triangleleft r"] \\
p \triangleleft q \arrow[r, "p \triangleleft g"'] & p \triangleleft q \triangleleft r
\end{tikzcd}$$

That is, do we have $f \mathbin{⨾} (p \triangleleft g) \overset{?}{=} g \mathbin{⨾} (f \triangleleft r)$? ◊

[a] When the name of an object is used in place of a morphism, we refer to the identity morphism on that object. So for instance, $f \triangleleft r$ is the composition product of f with the identity lens on r.

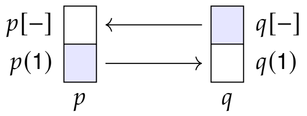
5.2 Lenses to composites

Lenses to composites—that is, lenses of the form $f\colon p \to q_1 \triangleleft q_2 \triangleleft \cdots \triangleleft q_n$ for some $n \in \mathbb{N}$ with composites as their codomains—will be ubiquitous in the remainder of our story. Fortunately, they have some very nice properties that make them convenient to work with. Before we explore these properties, we'll introduce a new way of visualizing lenses that is well-suited to capturing the behavior of lenses to composites.

5.2.1 Lenses as polyboxes

First, let us consider what goes into specifying a lens $f\colon p \to q$. To visualize this, we will introduce a new form of notation, which we will call *polyboxes*, that will generalize well to lenses to composites:

$f\colon p \to q$



(5.31)

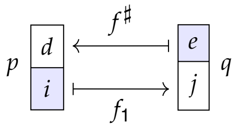
In our polyboxes, each pair of boxes stacked on top of each other represents a single polynomial. The bottom box in a pair can be filled with any position of the polynomial, while the top box must be filled with a direction of the polynomial at the position in the bottom box.

We think of (5.31) as depicting the lens f as a sort of gadget that acts like an automated spreadsheet: the blue boxes accept user input, while the white boxes are computed based on what is entered into the blue boxes according to the spreadsheet's preprogrammed rules. The arrows track the flow of information, starting from the bottom left.

When the user fills the bottom left blue box with a p-position i, the arrow to the right tells us that the gadget should automatically fill the bottom right white box with some q-position j, based on the value i that has already been entered. This process yields a map $i \mapsto j$ that corresponds to the on-position function f_1 of the lens.

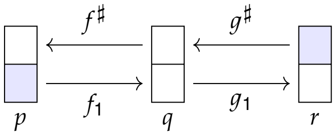
Then when the user fills the top right blue box with a $q[j]$-direction e, the arrow to the left tells us that the gadget should automatically fill the top left white box with some $p[i]$-position d, based on the values i and e that have already been entered. Fixing $i \in p(1)$, this process yields a map $e \mapsto d$ that corresponds to the on-directions function $f_i^\sharp$ of the lens.

So when both the user and the automation have finished filling all the boxes, we'll end up with something that looks like this:



Here, of course, $j \coloneqq f_1(i)$ and $d \coloneqq f_i^\sharp(e)$. So a lens is any protocol that will fill in the white boxes once the user fills in the blue boxes, following the directions of the arrows drawn.

If we have two composable lenses $f\colon p \to q$ and $g\colon q \to r$, then we can piece their polyboxes together to form polyboxes for their composite, $f \mathbin{⨟} g\colon p \to r$:

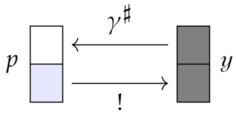


The bottom (position) box for q, which would normally be blue as part of the polyboxes for $g\colon q \to r$, is instead filled in via f_1; similarly, the top (direction box) for q, which would normally be blue as part of the polyboxes for $f\colon p \to q$, is filled in via $g^\sharp$. This forms a gadget that is equivalent to what the polyboxes would be for $f \mathbin{⨟} g$.

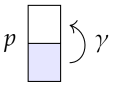
5.2.2 Situations as polyboxes

When a polynomial has only 1 position (i.e. it is representable), we shade its bottom (position) polybox gray to indicate that there is no choice to be made there, either by the user or by the automation; similarly, if a polynomial has only 1 direction at every position (i.e. it is linear), we shade its top (direction) polybox gray. So both polyboxes for y are shaded gray.

In Section 3.3.4, we discussed situations, which are lenses with codomain y. A situation $\gamma\colon p \to y$, then, can be depicted as follows, where ! is the unique function into 1:



But this is equivalent to a gadget where, when the user fills in the bottom left blue box with a p-position i, the top left white box is automatically filled with a $p[i]$-direction d. So we can redraw the gadget like so:



(5.32)

This lines up with what we already know: that a situation $\gamma\colon p \to y$ is just a dependent function $\gamma\colon (i \in p(1)) \to p[i]$.

5.2.3 Lenses to composites as polyboxes

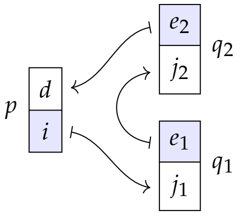
A lens $p \to q_1 \triangleleft q_2$ is an element of the set

$$\begin{aligned}\mathbf{Poly}(p, q_1 \triangleleft q_2) &\cong \mathbf{Poly}\left(p, \sum_{j_1 \in q_1(1)} \prod_{e_1 \in q_1[j_1]} \sum_{j_2 \in q_2(1)} \prod_{e_2 \in q_2[j_2]} y\right) && (5.6)\\ &\cong \prod_{i \in p(1)} \sum_{j_1 \in q_1(1)} \prod_{e_1 \in q_1[j_1]} \sum_{j_2 \in q_2(1)} \prod_{e_2 \in q_2[j_2]} p[i]. && (2.54)\end{aligned}$$

So we can write down the instructions for picking a lens $p \to q_1 \triangleleft q_2$ as follows.

```
To choose a lens $p \to q_1 \triangleleft q_2$:
  1. for each $p$-position $i$:
    1.1. choose a $q_1$-position $j_1$;
    1.2. for each $q_1[j_1]$-direction $e_1$:
      1.2.1. choose a $q_2$-position $j_2$;
      1.2.2. for each $q_2[j_2]$-direction $e_2$:
        1.2.2.1. choose a $p[i]$-direction $d$.
```

We could try to write out the dependent functions that these instructions correspond to. Alternatively, we could simply draw this protocol out using polyboxes, with every "for each" step corresponding to a user-maintained blue box and every "choose" step corresponding to an automated white box:



(5.33)

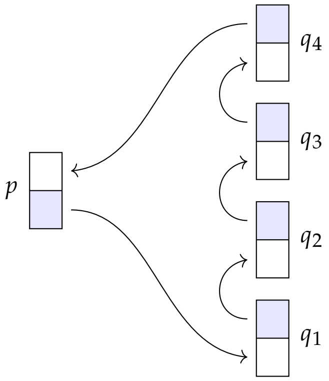
Whenever we draw two pairs of polyboxes on top of each other, as we do with the polyboxes for q_1 and q_2 above on the right, we are indicating that the entire column of polyboxes depicts the composite of the polynomials depicted by each individual pair. So the column of polyboxes on the right represents the composite $q_1 \triangleleft q_2$. In particular, the position in the bottom box of the top pair is the position associated with the direction in the top box of the bottom pair, for the depicted position of the composite.

So a lens $p \to q_1 \triangleleft q_2$ is any protocol that will fill in the white boxes above as the user fills in the blue boxes in the direction of the arrows. We'll see this in action in Example 5.34.

In fact, (5.32), (5.33), and (5.31) are the polybox depictions of the $n = 0, n = 1$, and $n = 2$ examples of lenses $p \to q_1 \triangleleft \cdots \triangleleft q_n$ to n-fold composites. In general, for any $n \in \mathbb{N}$, we can apply

$$\begin{aligned}\mathbf{Poly}(p, q_1 \triangleleft \cdots \triangleleft q_n) &\cong \mathbf{Poly}\left(p, \sum_{j_1 \in q_1(1)} \prod_{e_1 \in q_1[j_1]} \cdots \sum_{j_n \in q_n(1)} \prod_{e_n \in q_n[j_n]} y\right) && (5.6)\\ &\cong \prod_{i \in p(1)} \sum_{j_1 \in q_1(1)} \prod_{e_1 \in q_1[j_1]} \cdots \sum_{j_n \in q_n(1)} \prod_{e_n \in q_n[j_n]} p[i], && (2.54)\end{aligned}$$

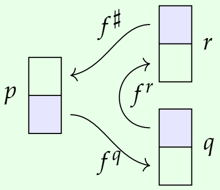
so the polybox depiction of $p \to q_1 \triangleleft \cdots \triangleleft q_n$ generalizes analogously. For example, here are the polyboxes corresponding to a lens to a 4-fold composite:



These lenses to n-fold composites lend themselves to a very natural interpretation in terms of our decision making language. Each of p's decisions is passed forward to a decision for q_1 to make. For every option that q_1 may choose, there is then also a decision for q_2 to make. Then for every option that q_2 may choose, there is a decision for q_3 to make, and so on, all the way until q_n has picked an option. Together, all the options that $q_1, \ldots, q_n$ chose then inform the option that p should make for its original decision.

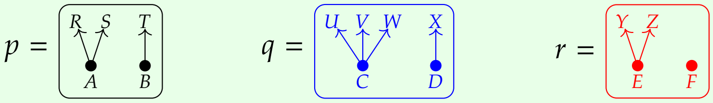
A lens $p \to q_1 \triangleleft \cdots \triangleleft q_n$ is a multi-step policy for p to make decisions by asking for decisions from q_1, then q_2, etc., all the way to q_n, and interpreting the results.

Example 5.34 (Lenses $p \to q \triangleleft r$). Consider a lens $f : p \to q \triangleleft r$. Let's label the three arrows in the lens's polybox depiction:

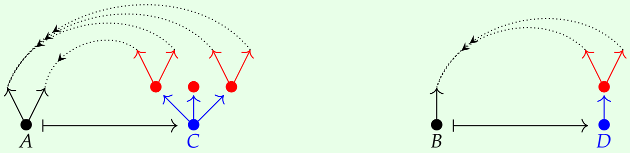


So the on-position function of f can be split into two parts: a function $f^q\colon p(1) \to q(1)$ and, for each $i \in p(1)$, a function $f^r_i\colon q[f^q(i)] \to r(1)$. Then the on-directions function $f^\sharp_i\colon (q \triangleleft r)[f_1(i)] \to p[i]$ takes the direction of q and the direction of r in the two blue boxes on the right and sends them to a direction of p at i to fill the white box on the left.

For example, let $p \coloneqq \{A\}y^{\{R,S\}} + By^{\{T\}}$, $q \coloneqq \{C\}y^{\{U,V,W\}} + \{D\}y^{\{X\}}$, and $r \coloneqq \{E\}y^{\{Y,Z\}} + \{F\}$.



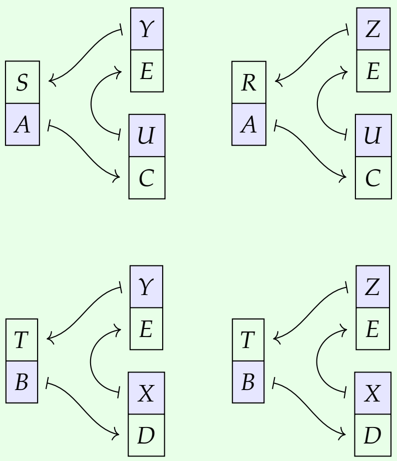
Here is a tree picture of a lens $f\colon p \to q \triangleleft r$:



If we write f as the corresponding triple $(f^q, f^r, f^\sharp)$, then we have

$$
\begin{gathered}
f^q(A) = C, \quad f^q(B) = D; \\
f^r_A(U) = E, \quad f^r_A(V) = F, \quad f^r_A(W) = E; \\
f^r_B(X) = E; \\
f^\sharp_A(U,Y) = S, \quad f^\sharp_A(U,Z) = R, \quad f^\sharp_A(W,Y) = R, \quad f^\sharp_A(W,Z) = R; \\
f^\sharp_B(X,Y) = T, \quad f^\sharp_B(X,Z) = T.
\end{gathered}
$$

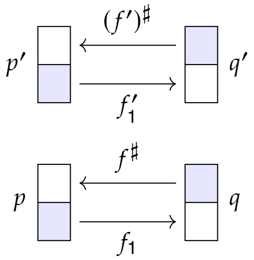
Polyboxes display the same data in a different format:



The third set of polyboxes, where the left blue box has been filled with an A and the bottom right blue box has been filled with a V, is worth highlighting: as $f^r_A(V) = F$, but $r[F] = \varnothing$, it is impossible to write a direction of r at F to go in the top right box. To indicate this, we color the top right box red and leave the arrow emerging from it dashsed.

5.2.4 The composition product of lenses as polyboxes

A special case of a lens whose codomain is a composite is a lens that is itself the composition product of lenses. If we draw such a lens as polyboxes by following the instructions from (5.18) and (5.19), we would really just be stacking the lenses on top of each other. For example, given lenses $f\colon p \to q$ and $f'\colon p' \to q'$, here is $f \triangleleft f'$ drawn as polyboxes:

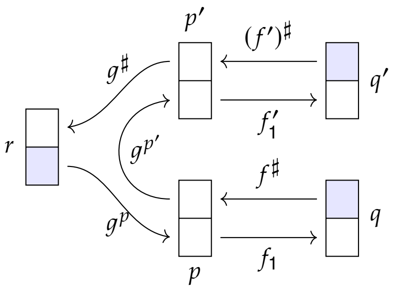


(5.35)

What differentiates this from simply writing down the polyboxes for f and the polyboxes for f' is that we are explictly associating the position that will fill the bottom box of p' with the direction that will fill the top box of p, and likewise the position that will fill the bottom box of q' with the direction that will fill the top box of q. Moreover, we have the user fill out the bottom set of boxes first and work their way up, so that, in particular, they can use the information they obtained from the behavior of f_1 and $f^\sharp$ to decide what to put in the bottom square of p'. So this really does depict a lens $p \triangleleft p' \to q \triangleleft q'$.

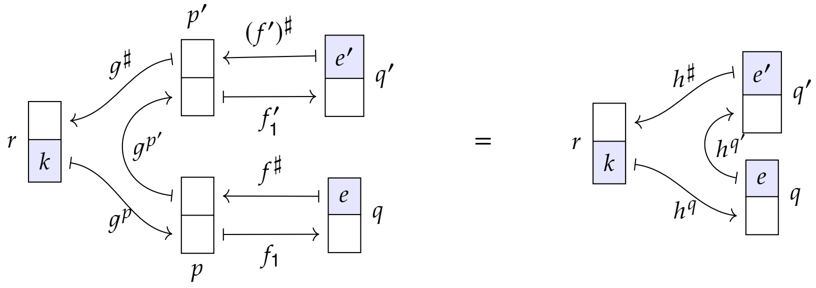
How does (5.35) relate to our usual polybox depiction of a lens to a composite, as in (5.33), but with the domain also replaced with a composite? A user who interacts with (5.35) could fill the bottom set of polyboxes for f independently of the top set of polyboxes for f'. Alternatively, along with what position goes in the bottom square of p, they could have already decided beforehand what position to put in the bottom square of p' for every possible direction that could end up in the top square of p. By (5.14), such a choice is equivalent to picking a position of the composite $p \triangleleft p'$. Then by (5.18), following just the bottom arrow f_1 leads to the corresponding position of q given by $f \triangleleft f'$, while filling in the top box of q and following $f^\sharp$ then f'_1 leads to the position of q' that goes in the bottom box of q'. Finally, once the user fills in the top box of q', following the top arrow $(f')^\sharp$ completes the specification of a direction of $p \triangleleft p'$. In this way, (5.35) can be thought of as a special case of (5.33).

We make a big deal out of it, but (5.35) really is just the polyboxes of two separate lenses drawn together. Where such polyboxes truly get interesting is when we compose them with polyboxes like (5.33). That is, given a lens $g \colon r \to p \triangleleft p'$, consider the polyboxes for $g \mathbin{⨟} (f \triangleleft f')$:



There's a lot going on with this lens! To fill out these polyboxes, we start from the bottom box of r, go all the way right to the bottom box of q, loop back left, up, and right again to the bottom box of q', then travel left all the way back to the top box of r.

Say we knew that $g \mathbin{⨟} (f \triangleleft f')$ were equal to some other lens $h \colon r \to q \triangleleft q'$:



Then we can read off the picture that

$$g^p \mathbin{⨟} f_1 = h^q,$$

that

$$f^\sharp_{g^p(k)} \mathbin{⨟} g^{p'}_k \mathbin{⨟} f'_1 = h^{q'}_k,$$

and that

$$(f')^\sharp_{g^{p'}(f^\sharp_{g^p(k)}(e))} \fatsemi g^\sharp_k = h^\sharp_k.$$

We will read equations off of polyboxes like this repeatedly in what follows.

5.3 Categorical properties of the composition product

We conclude this chapter by discussing several interesting properties of the composition product, many of which will come in handy in the following chapters.

5.3.1 Interaction with products and coproducts

It turns out that the composition product behaves well—albeit asymmetrically—with products and coproducts.

Proposition 5.36 (Left distributivity of $\triangleleft$ over $+$ and $\times$)**.** Given a polynomial r, the functor $(- \triangleleft r)\colon \mathbf{Poly} \to \mathbf{Poly}$ that sends each $p \in \mathbf{Poly}$ to $p \triangleleft r$ commutes with coproducts and products (up to natural isomorphism). That is, for any $p, q \in \mathbf{Poly}$, we have the following natural isomorphisms:

$$(p + q) \triangleleft r \cong (p \triangleleft r) + (q \triangleleft r) \tag{5.37}$$

and

$$pq \triangleleft r \cong (p \triangleleft r)(q \triangleleft r). \tag{5.38}$$

More generally, given a set A and polynomials $(p_a)_{a \in A}$, we have the following natural isomorphisms:

$$\left(\sum_{a \in A} p_a\right) \triangleleft r \cong \sum_{a \in A} (p_a \triangleleft r) \tag{5.39}$$

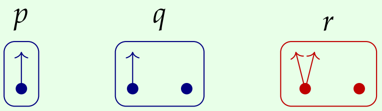
and

$$\left(\prod_{a \in A} p_a\right) \triangleleft r \cong \prod_{a \in A} (p_a \triangleleft r) \tag{5.40}$$

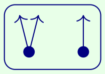
Proof. Formally, this comes down to the fact that (co)products of functors $\mathbf{Set} \to \mathbf{Set}$ are computed pointwise (Proposition 2.33) and that (co)products in $\mathbf{Poly}$ coincide with (co)products in $[\mathbf{Set}, \mathbf{Set}]$ (Propositions 2.50 and 2.79). One could instead give an explicit proof using (5.6); this is done in Exercise 5.41. In fact, we will see yet another proof of (5.40) (and thus (5.38)) in Exercise 5.50 #2. □

Exercise 5.41 (Solution here). Prove Proposition 5.36 using the explicit formula for $\triangleleft$ given in (5.6) by manipulating sums and products. ◊

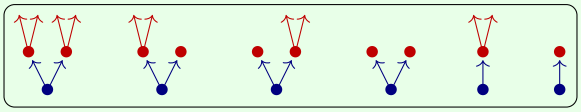
Example 5.42 (Picturing the left distributivity of $\triangleleft$ over $\times$). We want an intuitive understanding of the left distributivity given by (5.38). Let $p := y$, $q := y + 1$, and $r := y^2 + 1$, as shown here:



Then $pq \cong y^2 + y$ can be drawn as follows, with each corolla comprised of a corolla from p and a corolla from q with their roots glued together:

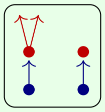
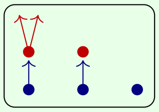
$pq \cong$ 

We can therefore draw $pq \triangleleft r$ by gluing corollas of r to leaves of pq in every way, as follows:

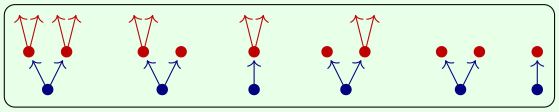
$pq \triangleleft r \cong$ 

So each tree in $pq \triangleleft r$ is obtained by gluing together the roots of a corolla from p and a corolla from q, then attaching corollas from r to each leaf.

Alternatively, we can compute $p \triangleleft r$ and $q \triangleleft r$ seperately, gluing corollas of r to leaves of p in every way, then to leaves of q in every way:

$p \triangleleft r \cong$  $q \triangleleft r \cong$ 

Their product is then obtained by taking each tree from $p \triangleleft r$ and pairing it with each tree from $q \triangleleft r$ by gluing their roots together:

$(p \triangleleft q)(p \triangleleft r) \cong$ 

So each tree in $(p \triangleleft r)(q \triangleleft r)$ is obtained by attaching corollas from r to each leaf of a corolla from p and a corolla from q before gluing their roots together.

But it doesn't matter whether we glue corollas from r to leaves first, or if we glue the roots of corollas from p and q together first–the processes are equivalent. Hence the isomorphism $pq \triangleleft r \cong (p \triangleleft r)(q \triangleleft r)$ holds.

Exercise 5.43 (Solution here). Follow Example 5.42 with coproducts (+) in place of products (×): use pictures to give an intuitive understanding of the left distributivity given by (5.37). ◊

Exercise 5.44 (Solution here). Show that for any set A and polynomials p, q, we have an isomorphism $A(p \triangleleft q) \cong (Ap) \triangleleft q$. ◊

In Section 5.3.2, we will see how to generalize the left distributivity of $\triangleleft$ over products to arbitrary limits. But first, we observe that right distributivity does not hold.

Exercise 5.45 (Solution here). Show that the distributivities of Proposition 5.36 do not hold on the other side:

1. Find polynomials p, q, r such that $p \triangleleft (qr) \not\cong (p \triangleleft q)(p \triangleleft r)$.
2. Find polynomials p, q, r such that $p \triangleleft (q + r) \not\cong (p \triangleleft q) + (p \triangleleft r)$. ◊

Nevertheless, there is something to be said about the relationship between $p \triangleleft q, p \triangleleft r$, and $p \triangleleft (qr)$. We'll see this in action after we discuss how $\triangleleft$ preserves limits on the left.

5.3.2 Interaction with limits

We saw in Theorem 4.32 that **Poly** has all limits, and we saw in Exercise 4.41 that these limits coincide with limits in $[\mathbf{Set}, \mathbf{Set}]$. Hence the argument in the proof of Proposition 5.36 can be generalized to arbitrary limits. It follows that $\triangleleft$ preserves all limits on the left. But we will present a proof of this fact from an alternative perspective: by appealing to the left coclosure of $\triangleleft$.

Proposition 5.46 (Meyers)**.** The composition product is left coclosed. That is, there exists a left coclosure operation, which we denote $\begin{bmatrix} - \\ - \end{bmatrix} : \mathbf{Poly}^{\mathrm{op}} \times \mathbf{Poly} \to \mathbf{Poly}$, such that there is a natural isomorphism

$$\mathbf{Poly}(p, q \triangleleft r) \cong \mathbf{Poly}\left(\begin{bmatrix} r \\ p \end{bmatrix}, q\right).$$

In particular, the left coclosure operation sends $r, p \in \mathbf{Poly}$ to

$$\begin{bmatrix} r \\ p \end{bmatrix} := \sum_{i \in p(1)} y^{r(p[i])}. \tag{5.47}$$

Proof. We have that

$$\begin{aligned} \mathbf{Poly}(p, q \triangleleft r) &\cong \prod_{i \in p(1)} q(r(p[i])) && (2.54) \\ &\cong \mathbf{Poly}\left(\sum_{i \in p(1)} y^{r(p[i])}, q\right) && (2.54) \\ &\cong \mathbf{Poly}\left(\begin{bmatrix} q \\ p \end{bmatrix}, r\right). && (5.47) \end{aligned}$$

□

Corollary 5.48 (Left preservation of limits)**.** Fix $r \in \mathbf{Poly}$. The functor $(- \triangleleft r)\colon \mathbf{Poly} \to \mathbf{Poly}$ that sends each $p \in \mathbf{Poly}$ to $p \triangleleft r$ preserves all limits (up to natural isomorphism). That is, for any category $\mathcal{J}$ and functor $p_{-}\colon \mathcal{J} \to \mathbf{Poly}$, there is a natural isomorphism

$$\left(\lim_{j\in\mathcal{J}} p_j\right) \triangleleft q \cong \lim_{j\in\mathcal{J}}(p_j \triangleleft q). \tag{5.49}$$

Proof. By Proposition 5.46, the functor $(- \triangleleft r)\colon \mathbf{Poly} \to \mathbf{Poly}$ is the right adjoint of the functor $\left[\begin{smallmatrix} r \\ - \end{smallmatrix}\right]\colon \mathbf{Poly} \to \mathbf{Poly}$, and right adjoints preserve limits. □

Exercise 5.50 (Solution here).

1. Complete Exercise 5.26 #3 using (5.49) and (5.39).
2. Deduce (5.40) using (5.49).

◊

A connected limit is one whose indexing category J is nonempty and connected. That is, J has at least one object, and any two objects are connected by a finite zigzag of arrows.

Example 5.51. The following categories are connected:

$$\boxed{\bullet} \qquad \boxed{\bullet \rightrightarrows \bullet} \qquad \boxed{\bullet \to \bullet \leftarrow \bullet} \qquad \boxed{\bullet \leftarrow \bullet \leftarrow \bullet \leftarrow \cdots}$$

In particular, equalizers, pullbacks, and directed limits are examples of connected limits.

The following categories are *not connected*:

$$\boxed{} \qquad \boxed{\bullet \quad \bullet} \qquad \boxed{\bullet \quad \bullet \to \bullet}$$

In particular, terminal objects and products are *not* examples of connected limits.

Theorem 5.52 (Preservation of connected limits)**.** The operation $\triangleleft$ commutes with connected limits in both variables. That is, if J is a connected category, $p\colon J \to \mathbf{Poly}$ is a functor, and $q \in \mathbf{Poly}$ is a polynomial, then there are natural isomorphisms

$$\left(\lim_{j\in J} p_j\right) \triangleleft q \cong \lim_{j\in J}(p_j \triangleleft q) \qquad \text{and} \qquad q \triangleleft \left(\lim_{j\in J} p_j\right) \cong \lim_{j\in J}(q \triangleleft p_j)$$

Sketch of proof. The claim for the left variable follows as in the proof of Proposition 5.36: limits of functors $\mathbf{Set} \to \mathbf{Set}$ are computed pointwise and $\mathbf{Poly}$ is a full subcategory of $\mathbf{Set}^{\mathbf{Set}}$ closed under limits. The claim for the right-hand variable comes down to the fact that polynomials are sums of representables; representable functors commute with all limits and sums commute with connected limits in **Set**. See [GK12, Proposition 1.6] for details. □

Exercise 5.53 (Solution here). Use Theorem 5.52 in the following.

1. Let p be a polynomial, thought of as a functor $p\colon \mathbf{Set} \to \mathbf{Set}$. Show that p preserves connected limits (of sets).
2. Show that for any polynomials p, q, r we have an isomorphism:

$$p \triangleleft (qr) \cong (p \triangleleft q) \times_{(p \triangleleft 1)} (p \triangleleft r)$$

◊

5.3.3 Interaction with parallel products

While we're here, it will be helpful to record the following.

Proposition 5.54. For any polynomial $q \in \mathbf{Poly}$, tensoring with q (on either side) preserves connected limits. That is, if J is connected and $p\colon J \to \mathbf{Poly}$ is a functor, then there is a natural isomorphism:

$$\left(\lim_{j \in J} p_j\right) \otimes q \cong \lim_{j \in J}(p_j \otimes q).$$

Proposition 5.55. For any polynomials p, p', q, q' there are natural maps

$$(p \triangleleft p') + (q \triangleleft q') \to (p + q) \triangleleft (p' + q') \tag{5.56}$$

$$(p \triangleleft p') \otimes (q \triangleleft q') \to (p \otimes pq) \triangleleft (p' \otimes q') \tag{5.57}$$

$$(p \triangleleft p') \times (q \triangleleft q') \leftarrow (p \times q) \triangleleft (p' \times q') \tag{5.58}$$

making $(+, \triangleleft)$ and $(\otimes, \triangleleft)$ duoidal structures and $(\times, \triangleleft)$ op-duoidal.

Proof. For (5.56) we have inclusion maps $p \to p + q$ and $p' \to p' + q'$, inducing a map $p \triangleleft p' \to (p + q) \triangleleft (p' + q')$. Similarly we obtain a map $q \triangleleft q' \to (p + q) \triangleleft (p' + q')$, so we get the desired map from the universal property of coproducts. It is straightforward to check that this is duoidal. The result for (5.58) is similar.

It remains to give a map (5.58).** □

5.3.4 Interaction with cartesian lenses

Proposition 5.59 ($\triangleleft$ preserves cartesian maps in both variables)**.** If $f\colon p \to p'$ and $g\colon q \to q'$ are cartesian then so is $(f \triangleleft g)\colon (p \triangleleft q) \to (p' \triangleleft q')$.

Proof. For any $h\colon A \to B$ of sets, all faces of the cube

$$\begin{tikzcd}
pqA \arrow[rr] \arrow[dr] \arrow[dd] & & pqB \arrow[dr] \arrow[dd] & \\
& pq'A \arrow[rr] \arrow[dd] & & pq'B \arrow[dd] \\
pq'A \arrow[rr] \arrow[dr] & & pq'B \arrow[dr] & \\
& p'q'A \arrow[rr] & & p'q'B
\end{tikzcd}$$

are pullbacks by Proposition 4.58 and Theorem 5.52. Hence the diagonal is too by standard properties of pullbacks. □

Exercise 5.60 (Solution here).

1. Show that if f is an isomorphism and g is vertical then $f \triangleleft g$ is vertical.
2. Find a polynomial q and a vertical morphism $f\colon p \to p'$ such that $(f \triangleleft \mathrm{id}_q)\colon (p \triangleleft q) \to (p' \triangleleft q)$ is not vertical. ◊

5.4 Exercise solutions

Solution to Exercise 5.8.

We are given $p \coloneqq y^2 + y^1$ and $q \coloneqq y^3 + 1$.

1. By standard polynomial multiplication, we have that $y^2 \triangleleft q \cong q \times q \cong y^6 + 2y^3 + 1$.
2. We have that $y^1 \triangleleft q \cong q \cong y^3 + 1$.
3. Combining the previous parts, we have that $(y^2 + y^1) \triangleleft q \cong q \times q + q \cong y^6 + 3y^3 + 2$.
4. Since $p[1] \cong 2$ and $q(1) \cong 2$, there are $2^2 = 4$ functions $p[1] \to q(1)$.
5. When $\bar{j}_1\colon p[1] \to q(1)$ is one of the two possible bijections, we have that
$$\sum_{d \in p[1]} q[\bar{j}_1(d)] \cong q[1] + q[2] \cong 3 + 0 \cong 3.$$
When $\bar{j}_1\colon p[1] \to q(1)$ sends everything to $1 \in q(1)$, we have that
$$\sum_{d \in p[1]} q[\bar{j}_1(d)] \cong q[1] + q[1] \cong 3 + 3 \cong 6.$$
Finally, when $\bar{j}_1\colon p[1] \to q(1)$ sends everything to $2 \in q(1)$, we have that
$$\sum_{d \in p[1]} q[\bar{j}_1(d)] \cong q[2] + q[2] \cong 0 + 0 \cong 0.$$
6. Since $p[2] \cong 1$ and $q(1) \cong 2$, there are $2^1 = 2$ functions $p[2] \to q(1)$.
7. When $j_2\colon p[2] \to q(1)$ maps to $1 \in q(1)$, we have that $\sum_{d \in p[2]} q[\bar{j}_2(d)] \cong q[1] \cong 3$, and when $\bar{j}_2\colon p[2] \to q(1)$ maps to $2 \in q(1)$, we have that $\sum_{d \in p[2]} q[\bar{j}_2(d)] \cong q[2] \cong 0$.
8. From the previous parts, we have that
$$\sum_{i \in p(1)} \sum_{\bar{j}_i\colon p[i] \to q(1)} y^{\sum_{d \in p[i]} q[j_i(d)]} \cong (2y^3 + y^6 + y^0) + (y^3 + y^0) \cong y^6 + 3y^3 + 2,$$
which agrees with what $p \triangleleft q$ should be.

Solution to Exercise 5.9.

1. Given representable polynomials $p \coloneqq y^A$ and $q \coloneqq y^B$, we have that $p \triangleleft q \cong \left(y^B\right)^A \cong y^{AB}$, which is also representable.
2. Given linear polynomials $p \coloneqq Ay$ and $q \coloneqq By$, we have that $p \triangleleft q \cong A(By) \cong ABy$, which is also linear.
3. Given constant polynomials $p \coloneqq A$ and $q \coloneqq B$, we have that $p \triangleleft q \cong A$, which is also constant (see also Exercise 5.26).

Solution to Exercise 5.12.

We wish to show that (5.13) could replace (5.11) in Definition 5.10. We claim that (5.11) and (5.13) are in fact the same function; that is, that the following square commutes:

$$\begin{tikzcd}
p(q(X)) \arrow[r, "f_{q(X)}"] \arrow[d, "p(g_X)"'] & p'(q(X)) \arrow[d, "p'(g_X)"] \\
p(q'(X)) \arrow[r, "f_{q'(X)}"'] & p'(q'(X))
\end{tikzcd}$$

Indeed it does, by the naturality of f.

Solution to Exercise 5.16.

1. The instructions associated with a polynomial $p \triangleleft p \triangleleft p$ are:
   1. choose a p-position i;
   2. for each $p[i]$-direction d:

      2.1. choose a p-position i';

      2.2. for each $p[i']$-direction d':

      2.2.1. choose a p-position i'';

      2.2.2. for each $p[i'']$-direction d'':

      2.2.2.1. choose a future.
2. To choose an element of $p \triangleleft p \triangleleft 1$:
   1. choose a p-position i;
   2. for each $p[i]$-direction d:

      2.1. choose a p-position i';

      2.2. for each $p[i']$-direction d':

      2.2.1. done.

Solution to Exercise 5.17.

We have lenses $f \colon p \to p'$ and $g \colon q \to q'$.

1. By Proposition 2.70, the $q(X)$-component of f is a function $f_{q(X)} \colon p(q(X)) \to p'(q(X))$ that sends every (i, h) with $i \in p(1)$ and $h \colon p[i] \to q(X)$ to $(f_1(i), f_i^\sharp \fatsemi h)$. We can think of the function $h \colon p[i] \to q(X)$ equivalently as a function $\bar{j}_i \colon p[i] \to q(1)$ and, for each $d \in p[i]$, a function $h_d \colon q[\bar{j}_i(d)] \to X$. So $f_{q(X)} \colon (p \triangleleft q)(X) \to (p' \triangleleft q)(X)$ sends
$$(i, \bar{j}_i, (h_d)_{d \in p[i]}) \mapsto \left(f_1(i), f_i^\sharp \fatsemi \bar{j}_i, \left(h_{f_i^\sharp(d')}\right)_{d' \in p'[f_1(i)]}\right).$$
2. By Proposition 2.70, the X-component of g is a function $g_X \colon q(X) \to q'(X)$ that sends every (j, k) with $j \in q(1)$ and $k \colon q[j] \to X$ to $(g_1(j), g_j^\sharp \fatsemi k)$ in $q'(X)$. Then by Proposition 2.46, applying p' to this X-component yields a function $p'(q(X)) \to p'(q'(X))$ that sends every $(i', \bar{j}'_{i'}, (h'_{d'})_{d' \in p'[i']})$ with $i' \in p'(1)$ as well as $\bar{j}'_{i'} \colon p'[i'] \to q(1)$ and $h'_{d'} \colon q[\bar{j}'_{i'}(d')] \to X$ to
$$\left(i', \bar{j}'_{i'} \fatsemi g_1, \left(g^\sharp_{\bar{j}'_{i'}(d')} \fatsemi h'_{d'}\right)_{d' \in p'[i']}\right).$$

3. By Definition 5.10, the horizontal composite of f and g is the natural transformation $f \mathbin{\triangleleft} g \colon p \mathbin{\triangleleft} p' \to q \mathbin{\triangleleft} q'$ whose X-component is the composite of the answers to #1 and #2, sending

$$\begin{aligned}(i, \bar{j}_i, (h_d)_{d \in p[i]}) &\mapsto \left(f_1(i), f_i^\sharp \fatsemi \bar{j}_i, \left(h_{f_i^\sharp(d')}\right)_{d' \in p'[f_1(i)]}\right) \\ &\mapsto \left(f_1(i), f_i^\sharp \fatsemi \bar{j}_i \fatsemi g_1, \left(g^\sharp_{\bar{j}_i(f_i^\sharp(d'))} \fatsemi h_{f_i^\sharp(d')}\right)_{d' \in p'[f_1(i)]}\right).\end{aligned}$$

4. We use Corollary 2.72 to translate the answer to #3 into a lens $f \mathbin{\triangleleft} g \colon p \mathbin{\triangleleft} q \to p' \mathbin{\triangleleft} q'$, as follows. Its on-positions function is the 1-component $(f \mathbin{\triangleleft} g)_1$, which sends every $(i, \bar{j}_i)$ with $i \in p(1)$ and $\bar{j}_i \colon p[i] \to q(1)$ to

$$(f_1(i), f_i^\sharp \fatsemi \bar{j}_i \fatsemi g_1).$$

Then for each such $(i, \bar{j}_i)$, if we apply the $(p \mathbin{\triangleleft} q)[(i, \bar{j}_i)]$-component of $f \mathbin{\triangleleft} g$ to the element $(i, \bar{j}_i, (\iota_d)_{d \in p[i]})$, where $\iota_d \colon q[\bar{j}_i(d)] \to (p \mathbin{\triangleleft} q)[(i, \bar{j}_i)] \cong \sum_{d \in p[i]} q[\bar{j}_i(d)]$ is the canonical inclusion, then take the last coordinate of the result, we obtain for each $d' \in p'[f_1(i)]$ the function

$$q'[g_1(\bar{j}_i(f_i^\sharp(d')))] \xrightarrow{g^\sharp_{\bar{j}_i(f_i^\sharp(d'))}} q[\bar{j}_i(f_i^\sharp(d'))] \xrightarrow{\iota_{f_i^\sharp(d')}} \sum_{d \in p[i]} q[\bar{j}_i(d)] \cong (p \mathbin{\triangleleft} q)[(i, \bar{j}_i)].$$

These can equivalently be thought of as a single function from

$$\sum_{d' \in p'[f_1(i)]} q'[g_1(\bar{j}_i(f_i^\sharp(d')))] \cong (p' \mathbin{\triangleleft} q')[(f \mathbin{\triangleleft} g)_1(i, \bar{j}_i)]$$

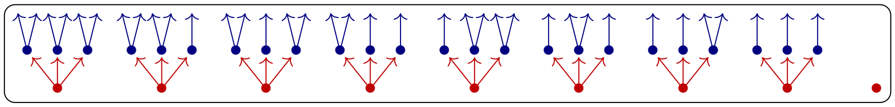
which Corollary 2.72 tells us is the on-directions function of $f \mathbin{\triangleleft} g$ at $(i, \bar{j}_i)$, that sends every (d', e') with $d' \in p'[f_1(i)]$ and $e' \in q'[g_1(\bar{j}_i(f_i^\sharp(d')))]$ to

$$\left(f_i^\sharp(d'), g^\sharp_{\bar{j}_i(f_i^\sharp(d'))}(e')\right).$$

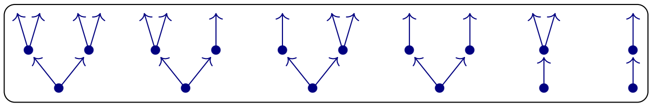
Solution to Exercise 5.24.

We have $p := y^2 + y$ and $q := y^3 + 1$ as in (5.20).

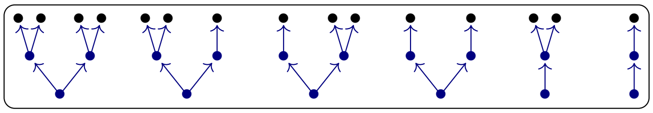
1. Here is a picture of $q \mathbin{\triangleleft} p$, where each tree is obtained by taking a corolla from q and gluing corollas from p to every leaf:



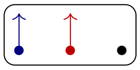
2. Here is a picture of $p \mathbin{\triangleleft} p$:



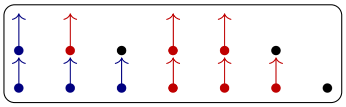
3. To obtain a picture of $p \mathbin{\triangleleft} p \mathbin{\triangleleft} 1$, we take our picture of $p \mathbin{\triangleleft} p$ and glue the single, leafless root from 1 to every (level-2) leaf:



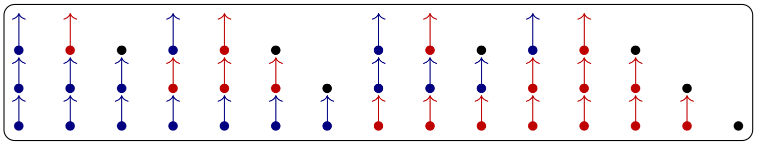
Now $r := 2y + 1$. Before we draw the composites, here's a picture of r itself, with different colors to distinguish the different positions:



4. Here is a picture of $r \triangleleft r$:

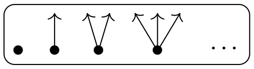


5. Here is a picture of $r \triangleleft r \triangleleft r$:

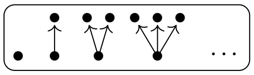


Solution to Exercise 5.26.

1. We pick the list polynomial, $p := 1 + y + y^2 + y^3 + \cdots$, drawn as follows:



Then here is a picture of $p \triangleleft 1$:



Below, X is a set and p is a polynomial.

2. A constant functor composed with any functor is still the same constant functor, so $X \triangleleft p \cong X$. We can also verify this using (5.6):

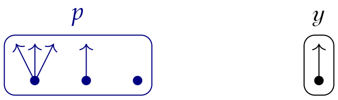
$$X \triangleleft p \cong \sum_{i \in X} \prod_{d \in \varnothing} \sum_{j \in p(1)} \prod_{e \in p[j]} y \cong \sum_{i \in X} 1 \cong X.$$

3. When viewed as functors, it is easy to see that $p \triangleleft X \cong p(X)$. We can also verify this using (5.6):

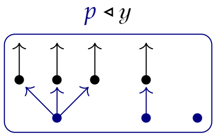
$$p \triangleleft X \cong \sum_{i \in p(1)} \prod_{d \in p[i]} \sum_{j \in X} \prod_{e \in \varnothing} y \cong \sum_{i \in p(1)} \prod_{d \in p[i]} \sum_{j \in X} 1 \cong \sum_{i \in p(1)} \prod_{d \in p[i]} X \cong \sum_{i \in p(1)} X^{p[i]} \cong p(X).$$

Solution to Exercise 5.27.

1. The polynomial y is the identity functor on **Set**.
2. Composing any functor with the identity functor yields the original functor, so $p \triangleleft y \cong p \cong y \triangleleft p$.
3. Before we draw $y \triangleleft p$ and $p \triangleleft y$, here are pictures of p and y individually as corolla forests:



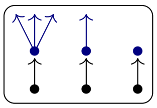
Now here is a picture of $p \triangleleft y$, obtained by gluing the one-leaf corolla of y to all the leaves of each of the corollas of p in turn:



This is just p with every direction extended up one level, so it is still a picture of p.

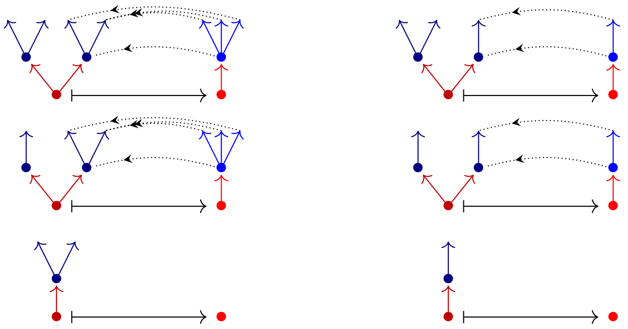
And here is a picture of $y \triangleleft p$, obtained by gluing each of the corollas of p to the single leaf of y:

$y \triangleleft p$



Solution to Exercise 5.29.

Using the definitions, instructions, and style from Example 5.28, we draw $g \triangleleft f \colon q \triangleleft p \to q' \triangleleft p'$:



Solution to Exercise 5.30.

Given arbitrary polynomials p, q, r and lenses $f \colon q \to p \triangleleft q$ and $g \colon q \to q \triangleleft r$, it is *not* necessarily the case that $f \fatsemi (p \triangleleft g) = g \fatsemi (f \triangleleft r)$! After all, we can let $p \coloneqq y$ and $q \coloneqq 2$ so that f is a lens $2 \to y \triangleleft 2 \cong 2$ (see Exercise 5.27) and g is a lens $2 \to 2 \triangleleft r \cong 2$ (see Exercise 5.26). Then by following the instructions for interpreting a composition product of lenses from either Exercise 5.17 or Example 5.28, we can verify that $p \triangleleft g = y \triangleleft g$ is a lens $2 \cong y \triangleleft 2 \to y \triangleleft 2 \triangleleft r \cong 2$ equivalent to the lens g, while $f \triangleleft r$ is a lens $2 \cong 2 \triangleleft r \to y \triangleleft 2 \triangleleft r \cong 2$ equivalent to the lens f. If, say, we let $f \colon 2 \to 2$ be the function sending everything to $1 \in 2$ and $g \colon 2 \to 2$ be the function sending everything to $2 \in 2$, then in this case $f \fatsemi (p \triangleleft g) = f \fatsemi g \neq g \fatsemi f = g \fatsemi (f \triangleleft r)$.

Solution to Exercise 5.41.

To prove Proposition 5.36, it suffices to verify (5.39) and (5.40), as (5.37) and (5.38) follow directly when $A \coloneqq 2$.

Given polynomials $(p_a)_{a \in A}$, recall that the position-set of the sum $\sum_{a \in A} p_a$ is $\sum_{a \in A} p_a(1)$, while the direction-set at each position (a, i) with $a \in A$ and $i \in p_a(1)$ is $p_a[i]$. So by (5.6), we have that

$$\begin{aligned}
\left(\sum_{a \in A} p_a\right) \triangleleft r &\cong \sum_{\substack{a \in A, \\ i \in p_a(1)}} \prod_{d \in p_a[i]} \sum_{j \in r(1)} \prod_{e \in r[j]} y \\
&\cong \sum_{a \in A} \sum_{i \in p_a(1)} \prod_{d \in p_a[i]} \sum_{j \in r(1)} \prod_{e \in r[j]} y \\
&\cong \sum_{a \in A} (p_a \triangleleft r).
\end{aligned}$$

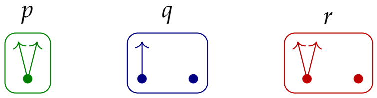
We can also recall that the position-set of the product $\prod_{a \in A} p_a$ is $\prod_{a \in A} p_a(1)$, while the direction-set at each position $\bar{i} \colon (a \in A) \to p_a(1)$ is $\sum_{a \in A} p_a[\bar{i}(a)]$. So by (5.6), we have that

$$\left(\prod_{a \in A} p_a\right) \triangleleft r \cong \sum_{\bar{i} \in \prod_{a \in A} p_a(1)} \prod_{\substack{a \in A, \\ d \in p_a[\bar{i}(a)]}} \sum_{j \in r(1)} \prod_{e \in r[j]} y$$

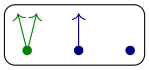
$$\cong \prod_{a\in A} \sum_{i\in p_a(1)} \prod_{d\in p_a[i]} \sum_{j\in r(1)} \prod_{e\in r[j]} y \qquad (2.28)$$
$$\cong \prod_{a\in A} (p_a \triangleleft r).$$

Solution to Exercise 5.43.

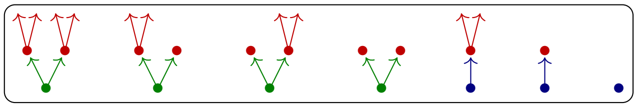
We want an intuitive understanding of the left distributivity of $\triangleleft$ over $+$. Let $p := y^2$, $q := y + 1$, and $r := y^2 + 1$, as shown here:



Then $p + q \cong y^2 + y + 1$ can be drawn as follows, consisting of all the corollas from p and q:

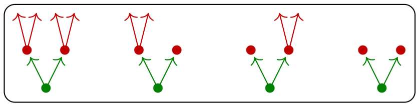
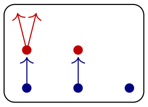
$p + q \cong$ 

We can therefore draw $(p + q) \triangleleft r$ by gluing corollas of r to leaves of $p + q$ in every way, as follows:

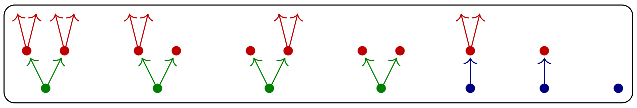
$(p + q) \triangleleft r \cong$ 

So each tree in $(p + q) \triangleleft r$ is obtained by taking either a corolla from p or a corolla from q, then attaching corollas from r to each leaf.

Alternatively, we can compute $p \triangleleft r$ and $q \triangleleft r$ seperately, gluing corollas of r to leaves of p in every way, then to leaves of q in every way:

$p \triangleleft r \cong$  $q \triangleleft r \cong$ 

Their coproduct then consists of all the trees from $p \triangleleft r$ and all the trees from $q \triangleleft r$:

$p \triangleleft r + q \triangleleft r \cong$ 

So each tree in $p \triangleleft r + q \triangleleft r$ is either a corolla from p with corollas from r attached to each leaf, or a corolla from q with corollas from r attached to each leaf.

But it doesn't matter whether we glue corollas from r to leaves first, or if we pool together corollas from p and q first–the processes are equivalent. Hence the isomorphism $(p + q) \triangleleft r \cong p \triangleleft r + q \triangleleft r$ holds.

Solution to Exercise 5.44.

Given a set A and polynomials p, q, the left distributivity of $\triangleleft$ over products from (5.40) implies that $(Ap) \triangleleft q \cong (A \triangleleft q)(p \triangleleft q)$, while Exercise 5.26 #3 implies that $A \triangleleft q \cong A$. So $(Ap) \triangleleft q \cong A(p \triangleleft q)$.

Solution to Exercise 5.45.

1. Let $p := y + 1, q := 1$, and $r := 0$. Then $p \triangleleft (qr) \cong (y + 1) \triangleleft 0 \cong 1$, while $(p \triangleleft q)(p \triangleleft r) \cong ((y + 1) \triangleleft 1)((y + 1) \triangleleft 0) \cong 2 \times 1 \cong 2$.
2. Again let $p := y + 1, q := 1$, and $r := 0$. Then $p \triangleleft (q + r) \cong (y + 1) \triangleleft 1 \cong 2$, while $(p \triangleleft q) + (p \triangleleft r) \cong ((y + 1) \triangleleft 1) + ((y + 1) \triangleleft 0) \cong 2 + 1 \cong 3$.

Solution to Exercise 5.50.

1. We wish to solve Exercise 5.26 #3 using (5.49) and (5.39). If we set $\mathcal{J}$ in (5.49) to be the empty category, then the limit of the functor from $\mathcal{J}$ is just the terminal object. It follows that $1 \triangleleft p \cong 1$. In other words, since $\triangleleft$ preserves limits on the left, and since terminal objects are limits, $\triangleleft$ preserves terminal objects on the left.

   Then a set X can be written as a sum $\sum_{x \in X} 1$, so by (5.39),

   $$X \triangleleft p \cong \left(\sum_{x \in X} 1\right) \triangleleft p \cong \sum_{x \in X} (1 \triangleleft p) \cong \sum_{x \in X} 1 \cong X.$$

2. If we set $\mathcal{J}$ in (5.49) to be the discrete category on the set A, then the limit of a functor from $\mathcal{J}$ is just an A-fold product, so (5.40) follows. In other words, since $\triangleleft$ preserves limits on the left, and since products are limits, $\triangleleft$ preserves products on the left.

Solution to Exercise 5.53.

1. Given a polynomial functor $p \colon \mathbf{Set} \to \mathbf{Set}$, we wish to show that p preserves connected limits of sets; that is, for a connected category J and a functor $X \colon J \to \mathbf{Set}$, we have

   $$p\left(\lim_{j \in J} X_j\right) \cong \lim_{j \in J} p(X_j).$$

   But we can identify **Set** with the full subcategory of constant functors in **Poly** and instead view X as a functor into **Poly**. Then by Exercise 5.26 #3, the left hand side of the isomorphism we seek is isomorphic to $p \triangleleft \left(\lim_{j \in J} X_j\right)$, while the right hand side is isomorphic to $\lim_{j \in J} \left(p \triangleleft X_j\right)$. These are isomorphic by Theorem 5.52.
2. Given $p, q, r \in \mathbf{Poly}$, we wish to show that $p \triangleleft (qr) \cong (p \triangleleft q) \times_{(p \triangleleft 1)} (p \triangleleft r)$. As 1 is terminal in **Poly**, the product qr can also be written as the pullback $q \times_1 r$. While products are not connected limits, pullbacks are, so by Theorem 5.52, they are preserved by precomposition with p. Hence the desired isomorphism follows.

Solution to Exercise 5.60.

1. **
2. **

Chapter 6

Polynomial comonoids

Imagine a sort of realm, where there are various positions you can be in. From every position, there are a number of moves you can make, possibly infinitely many. But whatever move you make, you'll end up in a new position. Well, technically it counts as a move to simply stay where you are, so you might end up in the same position. But wherever you move to, you can move again, and any number of moves from an original place counts as a single move. What sort of realm is this?

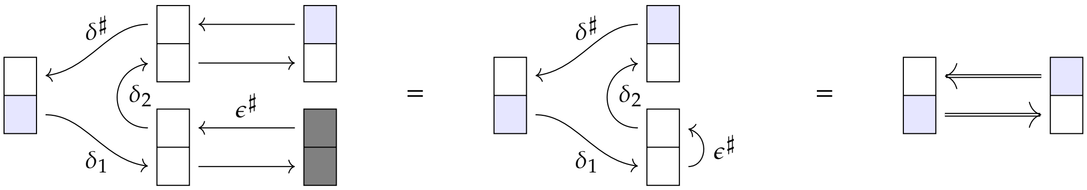
The most surprising aspects of **Poly** really begin with its comonoids. In 2018, researchers Daniel Ahman and Tarmo Uustalu showed that comonoids in $(\mathbf{Poly}, y, \triangleleft)$ can be identified with categories. Every category in the usual sense is a comonoid in **Poly** and every comonoid in **Poly** is a category. We find this revelation to be truly shocking, and it suggests some very different ways to think about categories. Let's go through it.

Definition 6.1 (Comonoid)**.** A *comonoid* in a monoidal category $(\mathcal{C}, y, \triangleleft)$ is a tuple (c, ϵ, δ) where $c \in \mathcal{C}$ is an object, and $\epsilon \colon c \to I$ and $\delta \colon c \to c \triangleleft c$ are maps, such that the following diagrams commute:

$$\begin{tikzcd}
y \triangleleft c \arrow[r, equal] & c \arrow[r, equal] \arrow[d, "\delta"] & c \triangleleft y \\
& c \triangleleft c \arrow[ul, "\epsilon \triangleleft c"] \arrow[ur, "c \triangleleft \epsilon"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
c \arrow[r, "\delta"] \arrow[d, "\delta"'] & c \triangleleft c \arrow[d, "c \triangleleft \delta"] \\
c \triangleleft c \arrow[r, "\delta \triangleleft c"'] & c \triangleleft c \triangleleft c
\end{tikzcd}
\tag{6.2}$$

We refer to a comonoid $P \coloneqq (p, \epsilon, \delta)$ in $(\mathbf{Poly}, y, \triangleleft)$ as a *polynomial comonoid*.

Here's a picture of one of the unit laws:



We'll put the associativity and the other unitality picture in the following exercise. The meaning of ϵ and δ will become clear; for those who want a hint, see **??**.

Exercise 6.3 (Solution here).

1. Draw the other unitality equation.
2. Draw the associativity equation. ◊

Example 6.4 (δ^n notation). Let (c, ϵ, δ) be a comonoid. From the associativity of δ, the two ways to get a map $c \to c \triangleleft c \triangleleft c$ have the same result. This is true for any $n \in \mathbb{N}$: we get an induced map $c \to c^{\triangleleft n+1}$, which by mild abuse of notation we denote δ^n:

$$c \xrightarrow{\delta} c \triangleleft c \xrightarrow{c \triangleleft \delta} c \triangleleft c \triangleleft c \xrightarrow{c^{\triangleleft 2} \triangleleft \delta} \cdots \xrightarrow{c^{\triangleleft n} \triangleleft \delta} c^{\triangleleft (n+1)}.$$

In particular, we have $\delta^1 = \delta$ and we may write $\delta^0 := \mathrm{id}_c$ and $\delta^{-1} := \epsilon$.

Polynomial comonoids are usually called *polynomial comonads*. Though polynomials p can be interpreted as polynomial *functors* $p\colon \mathbf{Set} \to \mathbf{Set}$, we do not generally emphasize this part of the story; we use it when it comes in handy, but generally we think of polynomials more as dependent arenas, or sets of corollas.

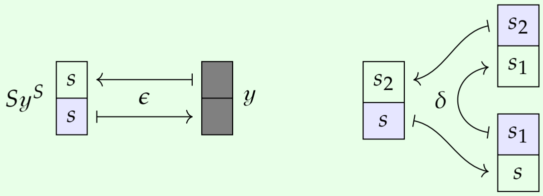
Example 6.5 (The state comonad Sy^S). Let S be a set, and consider the polynomial $p := Sy^S$. It has a canonical comonoid structure—often called the *state* comonad—as we discussed in **??**, page **??**. To say it in the current language, we first need to give maps $\epsilon\colon p \to y$ and $\delta\colon p \to p \triangleleft p$. By Eq. (5.6) and **??**, this is equivalent to giving functions

$$S \xrightarrow{\epsilon'} S \qquad\qquad S \xrightarrow{\delta'} \sum_{s' \in S} \prod_{s_1 \in S} \sum_{s_1' \in S} \prod_{s_2 \in S} S$$

We take ϵ' to be the identity and we take δ' to be

$$s \overset{\epsilon'}{\mapsto} s \qquad\qquad s \overset{\delta'}{\mapsto} (s' := s, s_1 \mapsto (s_1' := s_1, s_2 \mapsto s_2)). \tag{6.6}$$

If you know how to read such things, you'll see that each element $s \in S$ is just being passed in a straightforward way. But we find this notation cumbersome and prefer the poly-box notation.

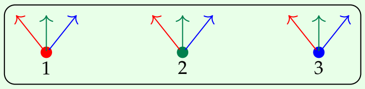


(6.7)

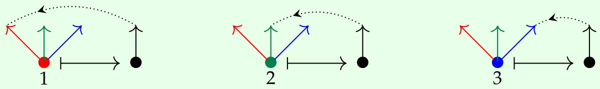
Exercise 6.8 (Solution here). Let $p := Sy^S$. For any $n \in \mathbb{N}$, write out the morphism of polynomials $\delta^n : p \to p^{\triangleleft(n+1)}$ either set-theoretically or in terms of poly-boxes as in Example 6.5 ◊

Example 6.9 (Picturing the comonoid Sy^S). Let's see this whole thing in pictures. First of all, let's take $S := 3 \cong \{\bullet, \bullet, \bullet\}$ and draw $\{\bullet, \bullet, \bullet\}y^{\{\bullet,\bullet,\bullet\}}$:

$3y^3 =$

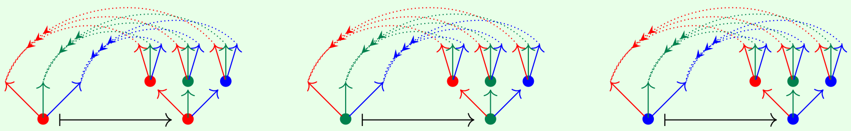


The map $\epsilon\colon Sy^S \to y$ can be drawn as follows:



It picks out one direction at each position, namely the one of the same color.

The map $\delta\colon Sy^S \to (Sy^S)^{\triangleleft 2}$ can be drawn as follows:



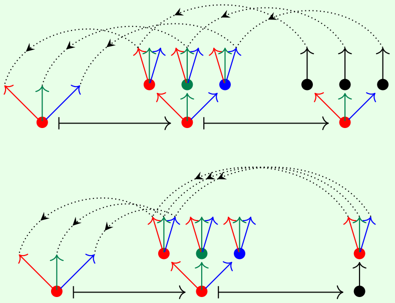
(6.10)

Note that $(Sy^S)^{\triangleleft 2}$ has SS^S, or in this case 81 many trees, only three of which are being pointed to by δ. That is, there is in general no rule on trees in that says the color of an arrow should agree in any sense with the color of the node it points to: (6.10) shows that the comonoid structure is pointing out the special trees where that does occur.

It remains to check the comonoid laws, the three commutative diagrams in (6.2). The first two say that the composites

$$Sy^S \xrightarrow{\delta} (Sy^S)^{\triangleleft 2} \xrightarrow{\mathrm{id}\triangleleft\epsilon} Sy^S \qquad \text{and} \qquad Sy^S \xrightarrow{\delta} (Sy^S)^{\triangleleft 2} \xrightarrow{\epsilon\triangleleft\mathrm{id}} Sy^S$$

are the identity. Let's return to the case $S = 3$. Then the second map in each case involves 81 different assignments, but only three of them will matter.[a] Since all three are strongly similar, we will draw only the red case. We also only draw the relevant passback maps.



We do not show associativity here, but instead leave it to the reader in Exercise 6.11.

[a] To say technically that we can disregard all but three positions in $(Sy^S)^{\triangleleft 2} \cong SS^S y^{SS}$, one can use Proposition 4.51.

Exercise 6.11 (Solution here). Let $S \coloneqq 2$ and $c \coloneqq 2y^2$.

1. Draw c using color if possible.
2. We know c is supposed to be the carrier of a comonoid (c, ϵ, δ). Which two maps $c \to c^{\triangleleft 3}$ are supposed to be equal by associativity?
3. Draw these two maps in the style of Example 6.9.
4. Are they equal? ◊

Exercise 6.12 (Monoid actions; solution here). Suppose that $(M, e, *)$ is a monoid, S is a set, and $\alpha \colon M \times S \to S$ is an M-action.

1. Show that Sy^M forms a comonoid.
2. Show that the projection $Sy^M \to y^M$ is a comonoid homomorphism.
3. M always acts on itself by multiplication. Is the associated comonoid structure on My^M the same or different from the one coming from Example 6.5? ◊

6.1 Speeding up dynamical systems

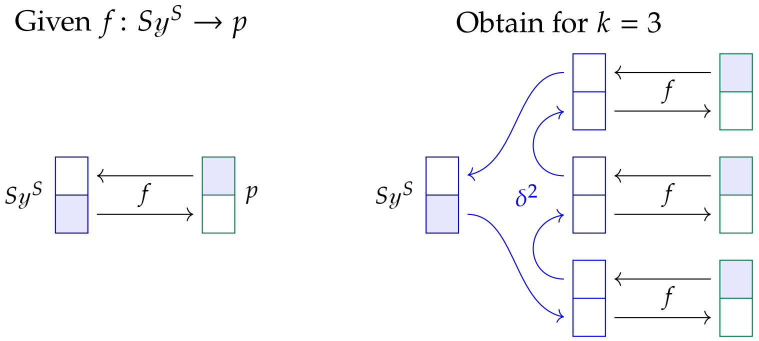
Suppose we have a dynamical system $f \colon Sy^S \to p$, and we want to make it go k-times faster. That is, in every moment, we want it to process k-many inputs, rather than one.

Since Sy^S has the structure of a comonoid, we know that for every $k \in \mathbb{N}$ we have a map $\delta^{k-1} \colon Sy^S \to (Sy^S)^{\triangleleft k}$ by Example 6.4. But we also have maps $f^{\triangleleft k} \colon (Sy^S)^{\triangleleft k} \to p^{\triangleleft k}$

because $\triangleleft$ is a monoidal product. Thus we can form the composite

$$\begin{tikzcd} Sy^S \arrow[r, "\delta^{k-1}"] \arrow[rr, bend right, "\mathrm{Spdup}_k(f)"'] & (Sy^S)^{\triangleleft k} \arrow[r, "f^{\triangleleft k}"] & p^{\triangleleft k} \end{tikzcd} \tag{6.13}$$

For every state $s \in S$, we now have a length-k strategy in p, i.e. a tree of height k in p, or explicitly a choice of p-position and, for every direction there, another p-position and so on k times. Here is a poly-box drawing for the $k = 3$ case:

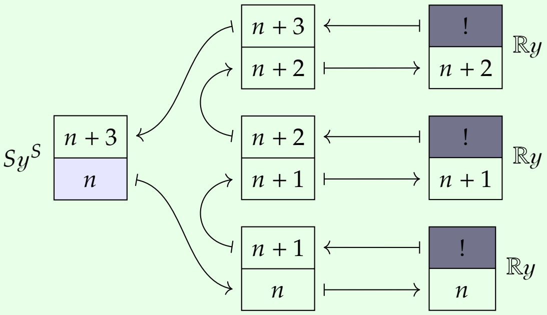


Example 6.14. Let $p := \mathbb{R}y^1$, let $S := \mathbb{N}$, and let $\binom{f^\sharp}{f_1}\colon Sy^S \to p$ be given by $f_1(n) := n$ and $f^\sharp(n, 1) := n + 1$. What is this speedup map $\mathrm{Spdup}_k(f)$? First of all, its type is

$$\mathrm{Spdup}_k(f)\colon Sy^S \to \mathbb{R}^k y,$$

meaning that it has the same set of states as before, but it outputs k-many reals in every moment.

So for example with $k = 3$ here is one moment of output:



So for example starting at initial state $n = 0$, we get the following output stream, e.g. for 4 seconds:

$$(0, 1, 2), (3, 4, 5), (6, 7, 8), (9, 10, 11).$$

6.2 Other comonoids

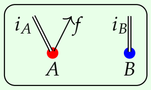
Once you know that these all important Sy^S-things are comonoids in **Poly**, it's interesting to ask "what are all the comonoids in **Poly**?" Let's discuss another one before answering the question in generality.

Example 6.15 (A simple comonoid that's not Sy^S). The polynomial $y^2 + y$ can be given a comonoid structure. Let's first associate names to its positions and directions.

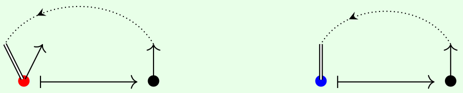
Define $w := \{A\}y^{\{i_A, f\}} + \{B\}y^{\{i_B\}}$; it is clearly isomorphic to $y^2 + y$, but its notation is meant to remind the reader of the walking arrow category

$$\mathcal{W} := \boxed{A \xrightarrow{f} B}$$

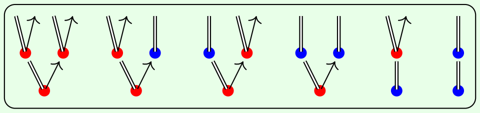
We will use the category $\mathcal{W}$ as inspiration for equipping w with a comonoid structure (w, ϵ, δ). The map ϵ will pick out identity arrows and the map δ will tell us about codomains and composition (which is rather trivial in the case of $\mathcal{W}$). Here's a picture of $w \cong y^2 + y$:

$w :=$ 

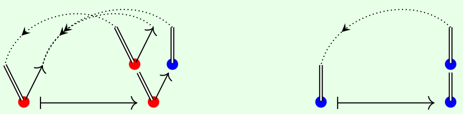
We first need to choose a map of polynomials $\epsilon\colon w \to y$; it can be identified with a dependent function $\epsilon^\sharp\colon (o \in w(1)) \to w[o]$, assigning to each position a direction there. Let's take $\epsilon^\sharp(A) := i_A$ and $\epsilon^\sharp(B) := i_B$:



Now we need a map of polynomials $\delta\colon w \to w \triangleleft w$. Let's draw out $w \triangleleft w$ to see what it looks like.

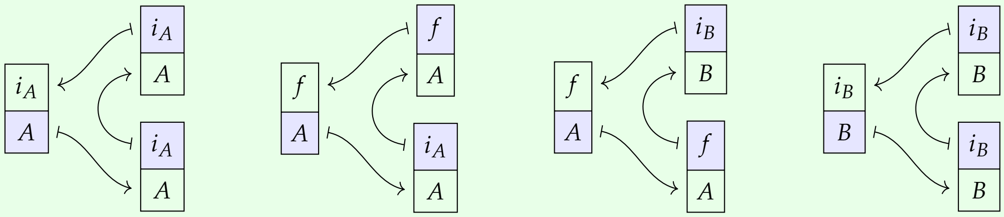
$w \triangleleft w =$ 

The map δ is going to tell us both about codomains and composition. Here it is:

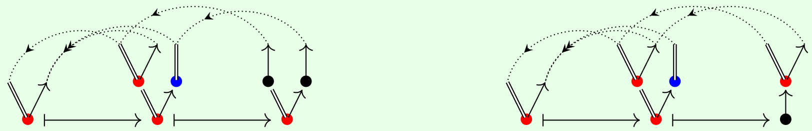


The on-positions map selects, for each position (either A or B) the two-level tree starting at that position and having the correct codomains: the identity arrow on A points to the corolla for A; the f map points to the corolla for B; and the identity arrow on B

points to the corolla for B. The on-directions maps assign the correct composites. Here is $\delta \colon w \to w \triangleleft w$ again, in terms of poly-boxes.



It remains to check that (w, ϵ, δ) really is a comonoid, i.e. that the diagrams in (6.2) commute. We will check unitality only for A; it is easier for B.



In both pictures, one can see that the composite map is the identity. We would do associativity here, but because the category $\mathcal{W}$ is so simple, associativity is guaranteed; this makes the pictures too trivial.

Exercise 6.16 (Solution here). Write out the data (c, ϵ, δ) for the comonoid corresponding to the category

$$\boxed{B \xleftarrow{f} A \xrightarrow{g} C}$$

For this exercise, you are not being asked to check the unitality or associativity conditions. ◊

Exercise 6.17 (Solution here). Show that if A is a set and $p \coloneqq Ay$ is the associated linear polynomial, then there exists a unique comonoid structure on p. ◊

Example 6.18 (The category of A-streams). For any set A, the set $A^{\mathbb{N}}$ of A-streams

$$s = (a_0 \rightsquigarrow a_1 \rightsquigarrow a_2 \rightsquigarrow a_3 \rightsquigarrow \cdots)$$

are the objects of a category, where the set of morphisms emanating from each stream $s \in A^{\mathbb{N}}$ is $\mathbb{N}$. The identity on s is given by 0 and the composite of two morphisms is the sum of the corresponding natural numbers.

We will see this category again in Example 7.13.

6.3 Comonoids in Poly are categories

It turns out that comonoids in **Poly** are precisely categories. Strangely, however, the a morphism between comonoids is not a functor but something people are calling a *cofunctor*.

Definition 6.19 (Cofunctor)**.** Let $\mathcal{C}$ be a category with object set C_0, morphism set C_1, $\mathrm{dom}, \mathrm{cod}\colon C_1 \to C_0$ the domain and codomain,[a] and similarly for $\mathcal{D}$. A *cofunctor* $F\colon \mathcal{C} \nrightarrow \mathcal{D}$ consists of

1. a function $F\colon C_0 \to D_0$ *on objects* and
2. a function $F^\sharp\colon C_0 \times_{D_0} D_1 \to C_1$ *backwards on morphisms*,

satisfying the following conditions:

i. $F^\sharp(c, \mathrm{id}_{F_0 c}) = \mathrm{id}_c$ for any $c \in C_0$;
ii. $F_0 \,\mathrm{cod}\, F^\sharp(c, g) = \mathrm{cod}\, g$ for any $c \in C_0$ and $g \in D_{F_0(c)}$;
iii. $F^\sharp(\mathrm{cod}\, F^\sharp(c, g_1), g_2) \circ F^\sharp(c, g_1) = F^\sharp(c, g_1 \mathbin{⨟} g_2)$ for composable arrows g_1, g_2 out of $F_0 c$.

In other words, $F^\sharp$ preserves identities, codomains, and compositions.

We denote by $\mathbf{Cat}^\sharp$ the category of categories and cofunctors.

[a] We privilege the domain function $\mathrm{dom}\colon C_1 \to C_0$ in the sense that an unnamed map $C_1 \to C_0$ will be assumed to be dom. For example, in the map $F^\sharp$, the pullback $C_0 \times_{D_0} D_1$ is of the diagram $C_0 \xrightarrow{F} D_0 \xleftarrow{\mathrm{dom}} D_1$.

The cofunctor laws can be written in commutative diagram form as follows:

$$\begin{tikzcd}
C_0 \times_{D_0} D_0 \arrow[r, "\cong"] \arrow[d, "\mathrm{id}_D"'] \arrow[dr, phantom, "\text{(i)}"] & C_0 \arrow[d, "\mathrm{id}_C"] \\
C_0 \times_{D_0} D_1 \arrow[r, "F^\sharp"'] & C_1
\end{tikzcd}
\qquad
\begin{tikzcd}
C_0 \times_{D_0} D_1 \arrow[r, "F^\sharp"] \arrow[d, "\pi_2"'] & C_1 \arrow[r, "\mathrm{cod}"] \arrow[d, phantom, "\text{(ii)}"] & C_0 \arrow[d, "F_0"] \\
D_1 \arrow[rr, "\mathrm{cod}"'] & & D_0
\end{tikzcd}$$

$$\begin{tikzcd}
C_0 \times_{D_0} D_1 \times_{D_0} D_1 \arrow[r, "⨟_D"] \arrow[d, "F^\sharp"'] & C_0 \times_{D_0} D_1 \arrow[r, "F^\sharp"] \arrow[d, phantom, "\text{(iii)}"] & C_1 \\
C_1 \times_{D_0} D_1 \arrow[r, "\cong"'] & C_1 \times_{C_0} C_0 \times_{D_0} D_1 \arrow[r, "F^\sharp"'] & C_1 \times_{C_0} C_1 \arrow[u, "⨟_C"']
\end{tikzcd}$$

Example 6.20 (Admissible sections). Consider the monoid $\mathcal{N} := (\mathbb{N}, 0, +)$, considered as a category with one object. For any category $\mathcal{C}$, a cofunctor $\varphi\colon \mathcal{C} \nrightarrow \mathcal{N}$ is called an *admissible section* [**Aguiar-thesis**]. We'll have more to say about these in Theorem 7.43, but our goal here is simply to unpack the definition.

To specify φ, we first say what it does on objects, but this is trivial: there is only one object in $\mathcal{N}$, so each object of $\mathcal{C}$ is sent to it. So the content of φ is all found in $\varphi^\sharp$, which assigns to each object $i \in \mathcal{C}$ and natural number $n \in \mathbb{N}$ a morphism $\varphi^\sharp(i, n)$ emanating from i. That seems like a lot of data, but we still have two laws to pare it down:

$$\varphi^\sharp(i,0) = i \qquad \text{and} \qquad \varphi^\sharp(i, n+n') = \varphi^\sharp(\varphi^\sharp(i,n), n').$$

Every natural number n is a sum of 1's, so if we denote $\varphi^\sharp(i,1)$ by $\phi(i)$, we in fact have

$$\varphi^\sharp(i,n) = \phi^{\circ n} i.$$

That is, $\varphi^\sharp(i,n)$ is just the n-fold application of the one-step case, ϕ.

Thus an admissible section of $\mathcal{C}$ is given by choosing, for each object $i \in \mathcal{C}$, a morphism $\phi(i)\colon i \to i'$ emanating from i.

Exercise 6.21 (Solution here). How many admissible sections does the category $\boxed{\bullet \to \bullet}$ have? ◊

Exercise 6.22 (Solution here). Let $\mathcal{Z} \coloneqq (\mathbb{Z}, 0, +)$ denote the monoid of integers and let $\mathcal{N}$ be that of natural numbers as above.

1. For a category $\mathcal{C}$, describe the data of a cofunctor $\mathcal{C} \nrightarrow \mathcal{Z}$.
2. What would you say is the canonical cofunctor $\mathcal{Z} \nrightarrow \mathcal{N}$?
3. Thinking of an admissible section $\mathcal{C} \nrightarrow \mathcal{N}$ as a policy, almost like a *physical law*, suppose it factors through $\mathcal{Z}$. Would you say that this lets you "run the law backwards"? ◊

Example 6.23 (Systems of ODEs). A system of ordinary differential equations (ODEs) in n variables, e.g.

$$\begin{aligned} \dot{x}_1 &= f_1(x_1, \ldots, x_n) \\ \dot{x}_2 &= f_2(x_1, \ldots, x_n) \\ &\vdots \\ \dot{x}_n &= f_n(x_1, \ldots, x_n) \end{aligned}$$

can be understood as a vector field on $\mathbb{R}^n$. Usually, we are interested in integrating this vector field to get flow lines, or integral curves. In other words, for each point $x = (x_1, \ldots, x_n)$ and each amount of time $t \in \mathbb{R}$, we can go forward from x for time t and arrive at a new point x^{+t}. These satisfy the equations

$$x^{+0} = x \qquad \text{and} \qquad x^{+t_1+t_2} = (x^{+t_1})^{+t_2}. \tag{6.24}$$

Let's call such things *dynamical systems* with time domain $(T, 0, +)$; above, we used $T = \mathbb{R}$, but any monoid will do.

Dynamical systems in the above sense are cofunctors $F\colon \mathbb{R}^n y^{\mathbb{R}^n} \nrightarrow y^T$. In order to say this, we first need to say how both $\mathcal{C} := \mathbb{R}^n y^{\mathbb{R}^n}$ and y^T are being considered as categories. The category $\mathcal{C}$ has objects $\mathbb{R}^n$, and for each object $x \in \mathbb{R}^n$ and outgoing arrow $v \in \mathbb{R}^n$, the codomain of v is $x + v$; in other words, v is a vector emanating from x. The identity is $v = 0$, and composition is given by addition. The category y^T is the monoid T considered as a category with one object, $\bullet$.

The cofunctor assigns to every object $x \in \mathbb{R}^n$ the unique object $F(x) = \bullet$, and to each element $t \in T$ the morphism $F^\sharp(x,t) = x^{+t} - x \in \mathbb{R}^n$, which can be interpreted as a vector emanating from x. Its codomain is $\operatorname{cod} F^\sharp(x,t) = x^{+t}$, and we will see that (6.24) ensures the cofunctoriality properties.

The codomain law ii is vacuously true, since y^T only has one object. Law i follows because $F^\sharp(x,0) = x^{+0} - x = 0$, and law iii follows as

$$F^\sharp(x^{+t_1}, t_2) + F^\sharp(x, t_1) = (x^{+t_1})^{+t_2} - x^{+t_1} + x^{+t_1} - x = x^{+t_1+t_2} - x = F^\sharp(x, t_1 + t_2).$$

Exercise 6.25 (Solution here).

1. Suppose that M, N are monoids (each is a category with one object). Are cofunctors between them related to monoid homomorphisms? If so, how?
2. Suppose $\mathcal{C}$ and $\mathcal{D}$ are categories and $F\colon \mathcal{C} \nrightarrow \mathcal{D}$ is a cofunctor. Does there necessarily exist a cofunctor $\mathcal{C}^{\mathrm{op}} \nrightarrow \mathcal{D}^{\mathrm{op}}$ that acts the same as F on objects? ◊

Theorem 6.26 (Ahman-Uustalu)**.** There is an equivalence of categories

$$\mathbf{Comon}(\mathbf{Poly}) \cong \mathbf{Cat}^\sharp.$$

Proof. This will be proved as Proposition 7.20 and Theorem 7.1. □

Our first goal is to understand how one translates between categories $\mathcal{C}$ and comonoids $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ in **Poly**. The idea is pretty simple: the objects of $\mathcal{C}$ are the positions of $\mathfrak{c}$

$$\mathrm{Ob}(\mathcal{C}) \cong \mathfrak{c}(1)$$

and for each such object i, the morphisms $\{f\colon i \to j \mid j \in \mathrm{Ob}(\mathcal{C})\}$ emanating from i in $\mathcal{C}$ are the directions $\mathfrak{c}[i]$ there.

Definition 6.27. Let $\mathcal{C}$ be a category. The *emanation polynomial for* $\mathcal{C}$ is the polynomial

$$\mathfrak{c} := \sum_{i \in \mathrm{Ob}(\mathcal{C})} y^{\sum_{j \in \mathrm{Ob}(\mathcal{C})} \mathcal{C}(i,j)}$$

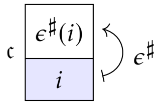
Exercise 6.28 (Solution here). What is the emanation polynomial for each of the following categories?

1. $\boxed{A \xrightarrow{f} B}$?
2. $\boxed{B \xleftarrow{f} A \xrightarrow{g} C}$?
3. The empty category?
4. The monoid $(\mathbb{N}, 0, +)$?
5. A monoid $(M, e, *)$?
6. The poset $(\mathbb{N}, \leq)$?
7. The poset $(\mathbb{N}, \geq)$? ◊

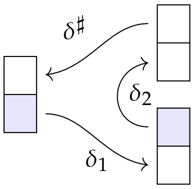
A category $\mathcal{C}$ is more than its emanation polynomial $\mathfrak{c}$, and a comonoid $(\mathfrak{c}, \epsilon, \delta)$ in **Poly** is more than its carrier polynomial $\mathfrak{c}$. The identities of $\mathcal{C}$ are all captured by the counit $\epsilon\colon \mathfrak{c} \to y$ and the codomain and composition information of $\mathcal{C}$ are all captured by the comultiplication map $\delta\colon \mathfrak{c} \to \mathfrak{c} \triangleleft \mathfrak{c}$. Our goal is to make this clear so that we can justly proclaim:

Comonoids in **Poly** *are precisely categories!*

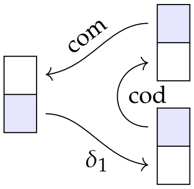
We want to understand how the counit ϵ and comultiplication δ in a comonoid $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ relate to identities, codomains, and composites in a category. We first use our work in **??** to get a better handle on ϵ and δ. For example, since $\epsilon\colon \mathfrak{c} \to y$ maps to the empty composite, we know by (5.32) that it is of the form



i.e. for every $i \in \mathfrak{c}(1)$, a choice of element $\epsilon^\sharp(i) \in \mathfrak{c}[i]$. Rather than call it $\epsilon^\sharp$, we will refer to this map as idy. Similarly, we know by (5.33) that δ is of the form

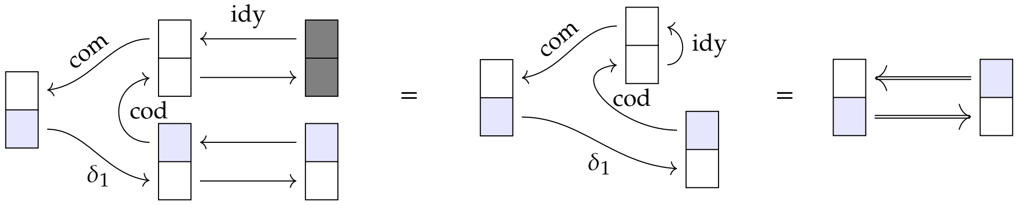


We've said that this secretly holds information about the codomains and composites for a category structure on $\mathfrak{c}$. How does that work? We will soon find that δ_1 is forced to be an identity, that δ_2 holds codomain information, and that $\delta^\sharp$ holds composite information. So we will use that notation here

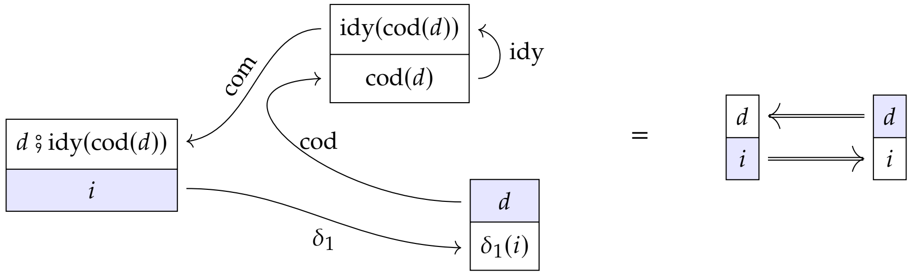


and our goal now is to see that the cod map really has something to do with codomains and that the com map really has something to do with composites, as advertised. What makes these true are the unitality and associativity equations required for $(\mathfrak{c}, \epsilon, \delta)$ to be a comonoid; see Definition 6.1.

To get started, we consider the first unitality equation from (6.2):

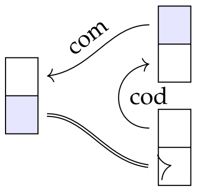


Let's add some arbitrary fillers $i \in \mathfrak{c}(1)$ and $d \in \mathfrak{c}[i]$ to the open slots, and hence obtain an equation:



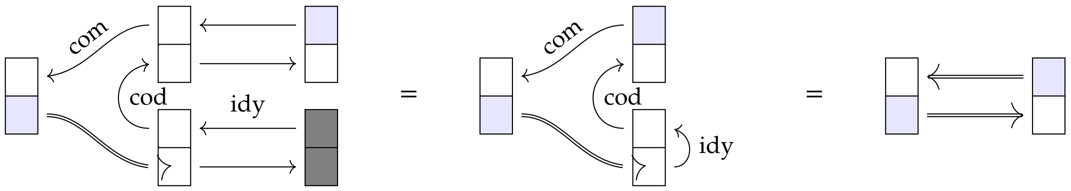
First it's saying that $\delta_1(i) = i$. This is great news; it means we can forget about δ_1, just as we said earlier. Second it's saying that $d \mathbin{\fatsemi} \mathrm{idy}(\mathrm{cod}(d)) = d$. Unpacking, this means that composing a morphism d with the identity morphism on its codomain returns d. It's neat to watch the comonoid laws declaring the standard laws of categories. It's like meeting a like-minded toad; we never knew toads could be like-minded, but the phenomena don't lie.

Before moving on, we redraw $\delta\colon c \to c \triangleleft c$ with the information-lacking δ_1 (which the first unitality equation said was always identity) and replace it with a double arrow:

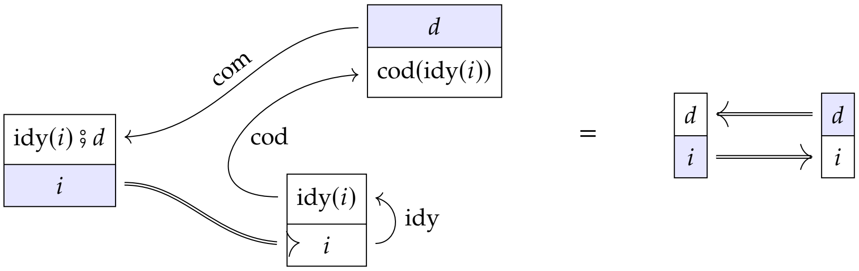


(6.29)

Now we can write the other unitality equation.

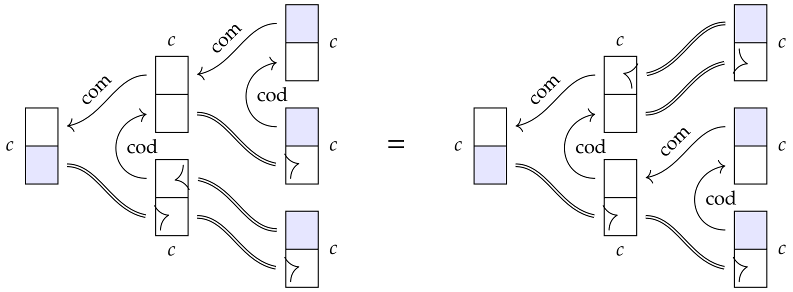


Let's add some arbitrary fillers $i \in c(1)$ and $d \in c[i]$ to get some equations:

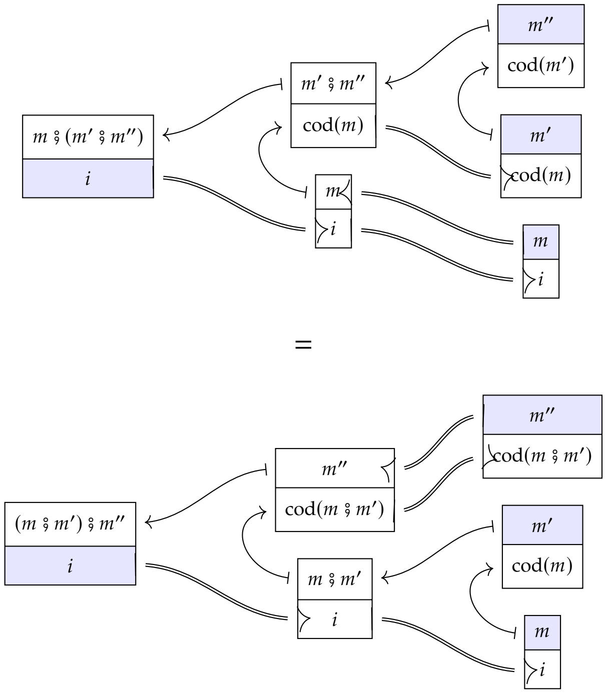


Ah, it's saying that it wants $\mathrm{cod}(\mathrm{idy}(i)) = i$, which makes sense—the codomain of the identity on i should be i—and that it wants $\mathrm{idy}(i) \fatsemi d = d$, i.e. that composing the identity on i with d should return d. We couldn't have said it better ourselves; thanks like-minded toad!

Finally we draw the associativity equation.



Let's fill it in with $i \in c(1)$ and a sequence $i \xrightarrow{m}\xrightarrow{m'}\xrightarrow{m''}$ of emanating morphisms:



Ah, it's saying that it wants $\mathrm{cod}(m') = \mathrm{cod}(m \fatsemi m')$; well yeah, that's how codomains should work. And it wants $m \fatsemi (m' \fatsemi m'') = (m \fatsemi m') \fatsemi m''$, classic associativity. Amazing; thanks again toad!

We've seen that all of the data and equations of categories are embedded, though in a very non-standard way, in the data and equations of polynomial comonoids.

Exercise 6.30 (Solution here). Let $\mathcal{C}$ be a category, $\mathfrak{c}$ its emanation polynomial, and $i \in \mathrm{Ob}(\mathcal{C})$ an object. This exercise is for people who know the definition of the coslice category $i/\mathcal{C}$ of objects under i. Is it true that there is an isomorphism

$$\mathrm{Ob}(c/\mathcal{C})i \cong^{?} \mathfrak{c}[i]$$

If so, describe it; if not, give a counterexample. ◊

6.4 Examples showing the correspondence between comonoids and categories

Example 6.31 (Monoids). Let $(M, e, *)$ be a monoid. Then we can construct a comonoid structure on the representable y^M. A morphism $y^M \to y$ can be identified with an element of M; under that identification we take $\epsilon := e$. Similarly, $y^M \triangleleft y^M \cong y^{M^2}$ and a morphism $y^M \to y^{M^2}$ can be identified with a function $M^2 \to M$; under that identification we take $\delta := *$.

Exercise 6.32 (Solution here). Finish Example 6.31 by showing that if $(M, e, *)$ satisfies the unitality and associativity requirements of a monoid in $(\mathbf{Set}, 1, \times)$ then (y^M, ϵ, δ) satisfies the unitality and associativity requirements of a comonoid in $(\mathbf{Poly}, y, \triangleleft)$. ◊

Example 6.33 (Monoid action). Suppose that $(M, e, *)$ is a monoid, S is a set, and $\alpha \colon M \times S \to S$ is an action. There is an associated category $\mathcal{A}$ with emanation polynomial Sy^M. In other words, it has objects set S and for every $s \in S$ there is an outgoing morphism for each $m \in M$, namely $s \xrightarrow{m} \alpha(m, s)$.

Exercise 6.34 (Solution here). With notation as in Example 6.33,

1. For a given object $s \in \mathcal{A}$, what is the identity morphism?
2. What is the composite of two morphisms in $\mathcal{A}$? ◊

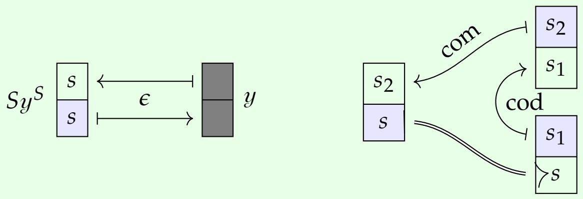
Example 6.35 (Cyclic lists). For any $n \in \mathbb{N}$, consider the monoid (group) $\mathbb{Z}/n$. As a functor $c_n := y^{\mathbb{Z}/n}$ sends a set X to the set of length-n tuples in X. But the comonoid structure lets us think of these as cyclic lists. Indeed, $\epsilon\colon c_n \to y$ allows us to pick out the "current" element via the map $\epsilon \triangleleft X\colon c_n \triangleleft X \to X$, and δ lets us move around the list.

We will see later that comonoids are closed under coproducts, so $\sum_{n\in\mathbb{N}} c_n$ is also a comonoid.

Example 6.36 (What category is Sy^S?). The first comonoid we introduced, back in Example 6.5 was $\mathscr{S} = (p, \epsilon, \delta)$, where $p = Sy^S$ for some set S. Now we know that comonoids correspond to categories. So what category $\mathcal{S}$ corresponds to $\mathscr{S}$?

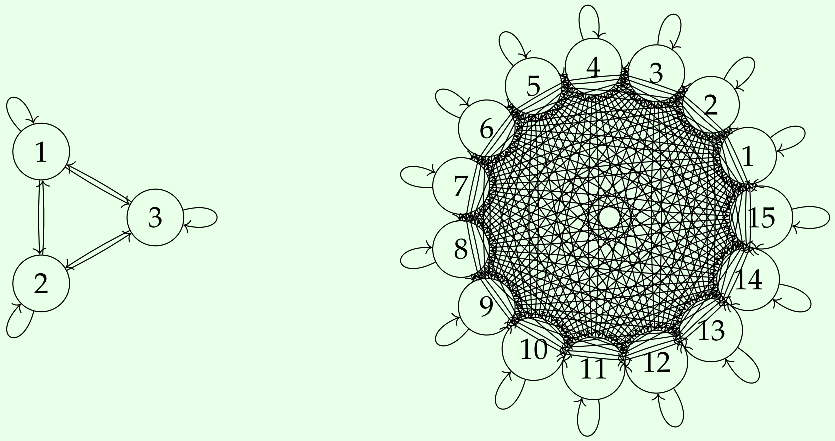
By the work above, we know that $\mathcal{S}$ has object set $S = p(1)$, and that for every object $s \in S$ there are S-many emanating morphisms, though we don't yet know their codomains nor the composition formula.

To calculate the codomains and compositions we examine the map $\delta\colon p \to p \triangleleft p$, which was given set-theoretically in (6.6) and in terms of poly-boxes in (6.7). We repeat it here for your convenience:



The ϵ map is saying that the identity on the object s is the emanating morphism s. Remember that both the set of objects and the set of morphisms emanating from any given object are S. The map cod $= \delta_2$ is telling us that the codomain of the morphism s_1 emanating from s is the object s_1, and that the composite of s_1 and s_2 is $s_1 \fatsemi s_2 = s_2$.

What this all means is that $\mathcal{S}$ is the category with S-many objects and a unique morphism $s \to s'$ for any $s, s' \in S$. Here are pictures for $S = 3$ and $S = 15$, with all maps (even identities) drawn:



Some people would call this the contractible groupoid, or the terminal category with S-elements, or the unique category whose underlying graph is complete on S vertices. The one that's *least* good for us will be "terminal" category, because as we'll see, we're going to be interested in different sorts of morphisms between categories than the usual ones, namely cofunctors rather than functors, and $\mathcal{S}$ is not terminal for cofunctors.

Anyway, to avoid confusion, we'll refer to $\mathcal{S}$ as the *state category on S*, because we will use these to think about states of dynamical systems, and also because the state comonad in functional programming is Sy^S.

Exercise 6.37 (Solution here). We showed in Exercise 6.17 that for any set A, the linear polynomial $p \coloneqq Ay$ has a unique comonoid structure. What category does it correspond to? ◊

Definition 6.38. Let $\mathcal{C}$ be a category and $c \in \mathrm{Ob}(\mathcal{C})$ an object. The *degree of c*, denoted $\deg(c)$, is the set of arrows in $\mathcal{C}$ that emanate from c.

If $\deg(c) \cong \mathsf{1}$, we say that c is *linear* and if $\deg(c) \cong \mathsf{n}$ for $n \in \mathbb{N}$, we say c has *degree n*.

Exercise 6.39 (Solution here).

1. If every object in $\mathcal{C}$ is linear, what does it mean about $\mathcal{C}$?
2. Is it possible for an object in $\mathcal{C}$ to have degree 0?
3. Find a category that has an object of degree $\mathbb{N}$.
4. How many categories are there that have just one linear and one quadratic (degree 2) object?
5. Is the above the same as asking how many comonoid structures on $y^2 + y$ there are?

◊

Exercise 6.40 (Solution here).

1. Find a category structure for the polynomial $y^{\mathsf{n}+1} + \mathsf{n}y$.
2. Would you call your category "star-shaped"? ◊

Exercise 6.41 (Solution here). Let S be a set. Is there any comonoid structure on Sy^S other than that of the state category? ◊

6.5 Morphisms of comonoids are cofunctors

Our next goal is to understand morphisms $\mathscr{C} \to \mathscr{D}$ between comonoids and what they look like as maps between categories $\mathcal{C} \to \mathcal{D}$.

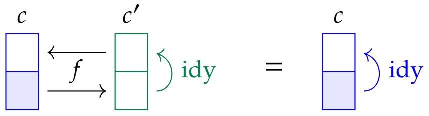
Cofunctors F are forward on objects, and backwards on morphisms. It's good to remember: Codomains are objects, so F preserves them going forwards; identities and composites are morphisms, so F preserves them going backwards.

Let's begin with a definition.

Definition 6.42 (Morphisms of comonoids)**.** Let $\mathscr{C} := (c, \epsilon, \delta)$ and $\mathscr{C}' := (c', \epsilon', \delta')$ be polynomial comonoids as in Definition 6.1. A *morphism* $\mathscr{C} \to \mathscr{C}'$ consists of a morphism $f : c \to c'$ of polynomials that commutes with the structure maps:

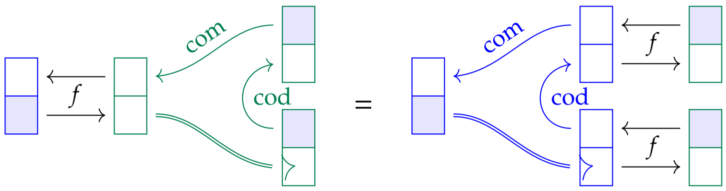
$$\begin{tikzcd}
c \arrow[r, "f"] \arrow[d, "\epsilon"'] & c' \arrow[d, "\epsilon'"] \\
y \arrow[r, equal] & y
\end{tikzcd}
\qquad
\begin{tikzcd}
c \arrow[r, "f"] \arrow[d, "\delta"'] & c' \arrow[d, "\delta'"] \\
c \triangleleft c \arrow[r, "f \triangleleft f"'] & c' \triangleleft c'
\end{tikzcd}
\tag{6.43}$$

Let's see the two laws of comonoid morphisms using poly-boxes. First the counit law:



(6.44)

Then the comultiplication law:



(6.45)

If we fill in Eq. (6.44) with an object $i \in \mathfrak{c}(1)$, we obtain the equation

$$f^\sharp(i, \mathrm{idy}(f_1(i))) = \mathrm{idy}(i),$$

which is the first law of Definition 6.19. If we fill in Eq. (6.45) with $i \in \mathfrak{c}(1)$ and $m \in \mathfrak{c}'[f(i)]$ and $m' \in \mathfrak{c}'[\mathrm{cod}(m)]$, we obtain the equations

$$\mathrm{cod}(m) = f_1\big(\,\mathrm{cod}\,\big(f^\sharp(i, m)\big)\big)$$
$$f^\sharp(i, \mathrm{com}(m, m')) = \mathrm{com}(f^\sharp(i, m), f^\sharp(\mathrm{cod}(f^\sharp(i, m)), m'))$$

and these are the second and third laws of Definition 6.19.

Exercise 6.46 (Solution here). Summarize the proof of Theorem 6.26, developed above. You may cite anything written in the text so far. ◊

Proposition 6.47. Let $F\colon \mathcal{C} \nrightarrow \mathcal{D}$ be a cofunctor, $c, c' \in \mathrm{Ob}(\mathcal{C})$ objects, and $g\colon F(c) \to F(c')$ a morphism in $\mathcal{D}$. Then if g is an isomorphism, so is $F^\sharp_c(g)$.

Proof. With $d := F(c)$, $d' := F(c')$, and g' the inverse of g, we have

$$\begin{aligned} \mathrm{id}_c &= F^\sharp_c(\mathrm{id}_d) \\ &= F^\sharp_c(g \mathbin{\fatsemi} g') \\ &= F^\sharp_c(g) \mathbin{\fatsemi} F^\sharp_{c'}(g') \end{aligned}$$

Thus $F^\sharp_c(g)$ is a section of $F^\sharp_{c'}(g')$. The opposite is true similarly, completing the proof. □

6.5.1 Examples of cofunctors

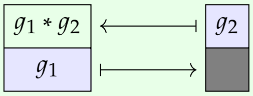
We saw in Theorem 6.26, summarized in Exercise 6.46, that cofunctors $\mathcal{C} \nrightarrow \mathcal{D}$ are the same thing as morphisms of comonoids $\mathscr{C} \to \mathscr{D}$, so we elide the difference. The question we're interested in now is: how do we think about cofunctors? What is a map of polynomial comonoids like?

The rough idea is that a cofunctor $\mathcal{C} \nrightarrow \mathcal{D}$ is, in particular, a morphism $\mathfrak{c} \to \mathfrak{d}$ in **Poly** between their emanation polynomials. This map preserves identities, codomains, and composition, which is great, but you still feel like you've got a map of polynomials on your hands: it goes forwards on objects and backwards on morphisms.

If a functor $\mathcal{C} \to \mathcal{D}$ is a picture *of $\mathcal{C}$ in $\mathcal{D}$, then a cofunctor $\mathcal{C} \nrightarrow \mathcal{D}$ is a D*-shaped crystallization of C.

Let's look at some examples to see how cofunctors look like crystallizations, or perhaps partitions.

Example 6.48. Let $(G, e, *)$ be a group and (y^G, ϵ, δ) the corresponding comonoid. There is a cofunctor $Gy^G \to y^G$ given by



To see this is a cofunctor, we check that identities, codomains, and compositions are preserved. For any g_1, the identity e is passed back to $g_1 * e = g_1$, and this is the identity on g_1 in Gy^G. Codomains are preserved because there is only one object in y^G. Composites are preserved because for any g_2, g_3, we have $g_1 * (g_2 * g_3) = (g_1 * g_2) * g_3$.

Exercise 6.49 (Solution here). Does the idea of Example 6.48 work when G is merely a monoid, or does something go subtly wrong somehow? ◊

Proposition 6.50. There is a fully faithful functor $\mathbf{Mon}^{\mathrm{op}} \to \mathbf{Cat}^{\sharp}$, whose image is precisely those categories whose emanation polynomial is representable.

Proof. Given a monoid $(M, e, *)$, we think of it as a category with one object; its emanation polynomial y^M is representable. A cofunctor between such categories carries no data in its on-objects part, and codomains are automatically preserved. Cofunctors $y^M \to y^N$ simply carry elements of N to elements of M, preserving identity and composition, exactly the description of monoid homomorphisms. □

Proposition 6.51. There is an adjunction

$$\mathbf{Cat}^{\sharp}(\mathcal{C}, Ay) \cong \mathbf{Set}(\mathrm{Ob}(\mathcal{C}), A)$$

where $\mathcal{C} \in \mathbf{Cat}^{\sharp}$ is a comonoid and $A \in \mathbf{Set}$ is a set.

Example 6.52. Consider the category $\mathbb{R}y^{\mathbb{R}}$, where the codomain of r emanating from x is $x + r$, identities are 0, and composition is given by addition. What are cofunctors into $\mathbb{R}y^{\mathbb{R}}$?

Let $\mathcal{C}$ be a category and $|\cdot|\colon \mathcal{C} \nrightarrow \mathbb{R}y^{\mathbb{R}}$ a cofunctor. It assigns to every object c both a real number $|c| \in \mathbb{R}$ and a choice of emanating morphism $|c|^{\sharp}(r)\colon c \to c_r$ such that $|c| + r = |c_r|$. This assignment satisfies some laws. Namely we have $c_0 = c$ and, given reals $r, s \in \mathbb{R}$, we have $(c_r)_s = c_{r+s}$.

Exercise 6.53 (Solution here).

1. Do you think a cofunctor $\mathcal{C} \nrightarrow \mathbb{R}y^{\mathbb{R}}$ as in Example 6.52 should be called an $(\mathbb{R}, 0, +)$-action on the objects of $\mathcal{C}$, or a filtration, or a valuation, or something else?
2. Why? ◊

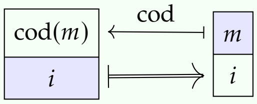
Exercise 6.54 (Solution here).

1. Over two discrete objects $\{A, B\}$, how many cofunctors

$$y^2 + y \cong \boxed{A \longrightarrow B} \nrightarrow \boxed{A \rightrightarrows B} \cong y^3 + y$$

are there from the walking arrow category to the walking parallel-arrows category?
2. What is meant more precisely by "over two discrete objects $\{A, B\}$" above? ◊

Exercise 6.55 (Solution here). Let $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ be a comonoid in **Poly**. We have a state category $\mathfrak{c}(1)y^{\mathfrak{c}(1)}$ on the set of objects of $\mathscr{C}$. There is a map of polynomials $\mathfrak{c}(1)y^{\mathfrak{c}(1)} \longrightarrow \mathfrak{c}$ given by



for an object $i \in \mathfrak{c}(1)$ and an outgoing morphism $m \in \mathfrak{c}[i]$. Is this map a cofunctor? ◊

Example 6.56 (Canonical cofunctors from state categories). Let $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ be a comonoid, where $\delta = (\text{id}, \text{cod}, \text{com})$ as in (6.29). For any position $i \in \mathfrak{c}(1)$, there is a cofunctor

$$(\text{cod}, \text{com})\colon \mathfrak{c}[i]y^{\mathfrak{c}[i]} \nrightarrow \mathfrak{c}.$$

That is, an object $f \in \mathfrak{c}[i]$ is also a morphism in $\mathcal{C}$ and we send it to its codomain $\text{cod}(f)$. A morphism in $\mathcal{C}$ emanating from $\text{cod}(f)$ is passed back to its composite with f.

Exercise 6.57 (Solution here). Suppose $\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$ is a comonoid.

1. Show that the map $(\text{cod}, \text{com})\colon \mathfrak{c}[i]y^{\mathfrak{c}[i]} \nrightarrow \mathfrak{c}$ from Example 6.56 satisfies the conditions necessary for being a cofunctor (identities, codomains, and composites).
2. Find a comonoid structure on the polynomial $p := \sum_{i \in \mathfrak{c}(1)} \mathfrak{c}[i]y^{\mathfrak{c}[i]}$ and a cofunctor $p \longrightarrow \mathfrak{c}$.
3. Is the polynomial map $p \longrightarrow \mathfrak{c}$ an epimorphism? ◊

Exercise 6.58 (Solution here). Suppose $\mathfrak{c}, \mathfrak{d}, \mathfrak{e}$ are polynomials, each with a comonoid structure, and that $f\colon \mathfrak{c} \to \mathfrak{d}$ and $g\colon \mathfrak{d} \to \mathfrak{e}$ are maps of polynomials.

1. If f and $f \mathbin{⨾} g$ are each cofunctors, is g automatically a cofunctor? If so, sketch a proof; if not sketch a counterexample.
2. If g and $f \mathbin{⨾} g$ are each cofunctors, is f automatically a cofunctor? If so, sketch a proof; if not sketch a counterexample. ◊

Exercise 6.59 (Solution here).

1. For any category $\mathcal{C}$ with emanation polynomial $\mathfrak{c}$, find a category with emanation polynomial $\mathfrak{c}y$.
2. Show your construction is functorial; i.e. given a cofunctor $\mathfrak{c} \nrightarrow \mathfrak{d}$, find one $\mathfrak{c}y \nrightarrow \mathfrak{d}y$, preserving identity and composition.
3. Is your functor either a monad or a comonad on $\mathbf{Cat}^{\sharp}$?
4. What category do you get by repeatedly applying this functor to y? ◊

Exercise 6.60 (Solution here). Are cofunctors between posets interesting?

1. Consider the chain poset $[n] \cong \sum_{i=1}^{n} y^i$. How many cofunctors are there from $[m] \to [n]$ for all $m, n \in \{0, 1, 2, 3\}$?
2. What does a cofunctor from y into a poset represent? Is there anything you'd call "asymmetric" about it? ◊

Exercise 6.61 (Solution here).

1. What is the finite set $\{\mathscr{Q}_1, \ldots, \mathscr{Q}_n\}$ of comonoids (defined up to isomorphism) for which the carrier polynomial is $y^2 + y$?
2. For each category $\mathscr{Q}_i$, describe how to imagine a cofunctor $\mathscr{C} \to \mathscr{Q}_i$ from an arbitrary category into it.
3. What cofunctors exist between the various $\mathscr{Q}_i$? ◊

Exercise 6.62 (Solution here). Let S be a set. Describe a way to visualize cofunctors from categories into the state category Sy^S. Feel free to focus on the case where S is a small finite set. Hint: use Proposition 6.47. ◊

Exercise 6.63 (Solution here).

1. Recall the star-shaped category $y^{\mathsf{n}+1} + \mathsf{n}y$ from Exercise 6.40. Describe cofunctors into it.
2. Describe cofunctors into Ay for a set A.
3. Describe cofunctors into $(\mathbb{N}, \leq)$.

4. Describe cofunctors into $(\mathbb{N}, \geq)$.
5. Let $y^4 + 2y^2 + y$ denote the commutative square category. List the cofunctors from it to the walking arrow category $y^2 + y$? There should be six or so. ◊

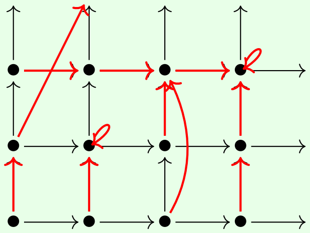
Example 6.64 (Objects aren't representable in $\mathbf{Cat}^\sharp$). For categories and ordinary functors, there is a category $\mathcal{T}$ that *represents objects*, in the sense that functors $\mathcal{T} \to \mathcal{C}$ are the same as objects in $\mathcal{C}$; indeed, take $\mathcal{T} = \boxed{\bullet}$ to be the terminal (one morphism) category.

This does not work for cofunctors, as we'll see in Exercise 6.65. The comonoid corresponding to $\mathcal{T}$ is y with its unique comonoid structure. Cofunctors $\mathcal{T} \nrightarrow \mathcal{C}$ are somewhat strange beasts: they can be identified with objects $c \in \mathcal{C}$ for which the codomain of every emanating morphism $c \to c'$ is $c' = c$ itself. The reason is the codomain condition (Definition 6.19, condition 2).

Exercise 6.65 (Solution here). We saw in Exercise 6.17 that $2y$ has a unique comonoid structure.

1. Show that for any category $\mathcal{T}$, there are $2^{\#\mathrm{Ob}(\mathcal{T})}$-many cofunctors $\mathcal{T} \nrightarrow 2y$.
2. Use the case of $\mathcal{C} := 2y$ to show that if a category $\mathcal{T}$ is going to represent objects as in Example 6.64 then $\mathcal{T}$ must have one object.
3. Now use a different $\mathcal{C}$ to show that if a category $\mathcal{T}$ is going to represent objects, it must have more than one object. ◊

Example 6.66 (Policies are co-representable). For a category $\mathcal{C}$, let's say that a *policy in* $\mathcal{C}$ is a choice, for each object $c \in \mathcal{C}$, of an emanating morphism $f : c \to c'$. For example, consider the category $(\mathbb{N}, \leq) \times (\mathbb{N}, \leq)$:



In red we have drawn a policy: every object has been assigned an emanating morphism to another object; there doesn't need to be any rhyme or reason to our choice.

For any category $\mathcal{C}$, the set of trajectories in $\mathcal{C}$ is in bijection with the set of cofunctors

$$\mathcal{C} \nrightarrow \mathcal{n}$$

where $\mathcal{n} = y^{\mathbb{N}}$ is the monoid of natural numbers under addition.

Exercise 6.67 (Solution here). At the end of Example 6.66 we said that a policy on $\mathcal{C}$ can be identified with a cofunctor $F\colon \mathcal{C} \nrightarrow \mathcal{N}$. But at first it appears that F includes more than just a policy: for every object $c \in \mathrm{Ob}(\mathcal{C})$ and natural number $n \in \mathbb{N}$, we have a morphism $F^\sharp_c(n)$ emanating from c. That's infinitely many emanating morphisms per object, whereas a policy seems to include only one emanating morphism per object.

Explain why looks are deceiving in this case: why is a policy on $\mathcal{C}$ the same as a cofunctor $\mathcal{C} \nrightarrow \mathcal{N}$? ◊

We will see later in Proposition 7.44 that the trajectories on a category form a monoid, and that this operation $\mathbf{Cat}^\sharp \to \mathbf{Mon}^{\mathrm{op}}$ is functorial and in fact an adjoint.

Exercise 6.68 (Continuous trajectories; solution here). Suppose we say that a continuous policy in $\mathcal{C}$ is a cofunctor $\mathcal{C} \nrightarrow \mathcal{R}$, where $\mathcal{R}$ is the monoid of real numbers under addition, considered as a category with one object.

Describe continuous trajectories in $\mathcal{C}$ using elementary terms, i.e. to someone who doesn't know what a cofunctor is and isn't yet ready to learn. ◊

Exercise 6.69 (Solution here). Let $\mathbb{R}/\mathbb{Z} \cong [0, 1)$ be the quotient of $\mathbb{R}$ by the $\mathbb{Z}$-action sending $(r, n) \mapsto r + n$. More down to earth, it's the set of real numbers between 0 and 1, including 0 but not 1.

1. Find a comonoid structure on $(\mathbb{R}/\mathbb{Z})y^{\mathbb{R}}$.
2. Is it a groupoid? ◊

Exercise 6.70 (Solution here).

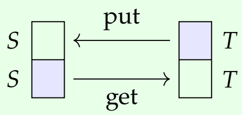
1. If two categories are isomorphic in **Cat**, does that imply they are isomorphic in $\mathbf{Cat}^\sharp$?
2. If so, prove it; if not, give a counterexample.
3. Is it true that for any two categories $\mathcal{C}, \mathcal{D}$, there is a bijection between the set of isomorphisms $\mathcal{C} \xrightarrow{\cong} \mathcal{D}$ in **Cat** and the set of isomorphisms $\mathcal{C} \overset{\cong}{\nrightarrow} \mathcal{D}$ between them in $\mathbf{Cat}^\sharp$?
4. If so, prove it; if not, give a counterexample. ◊

6.5.2 Very well behaved lenses

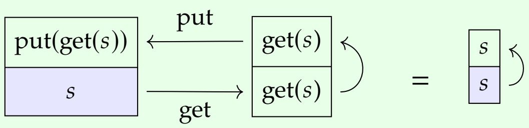
In the functional programming community, there is an important notion of very well-behaved lenses. These turn out to be precisely the cofunctors between state categories. Since state categories Sy^S play an important role in our theory, we take a bit of time to consider cofunctors between them.

Example 6.71 (Very well-behaved lenses). Recall from Example 6.5 that for any set S, we have the "state" category with emanation polynomial Sy^S. What are the comonoid morphisms—cofunctors—between different state categories?

First, such a comonoid morphism includes a morphism of polynomials $f\colon Sy^S \to Ty^T$; we'll use the standard terminology of "get" and "put":



Let's apply the unit-homomorphism property (6.44)



It says that put(get(s)) = s. This is typically called the get-put law.

We leave the comultiplication-homomorphism law to Exercise 6.72, where we will see that it specifies that get and put must satisfy two other properties, called the put-put and the put-get laws.

Exercise 6.72 (Solution here). Complete Example 6.71.

1. Write out the comultiplication law from (6.44) in terms of poly-boxes.
2. What set-theoretic equations are forced by the comultiplication law?
3. Can you see why they might be called put-put and put-get? ◊

Example 6.73 (Very well-behaved lenses are kinda boring). We saw in Exercise 6.72 that a comomonoid homomorphism (cofunctor) $Sy^S \nrightarrow Ty^T$ between state comonoids can be characterized as a pair of functions get: $S \to T$ and put: $S \times T \to S$ satisfying get-put, put-get, and put-put.

In fact, it turns out that this happens if and only if get is a product projection! For example, if the cardinalities $|S|$ and $|T|$ of S and T are finite and $|S|$ is not divisible by $|T|$, then there are no cofunctors $Sy^S \nrightarrow Ty^T$. A stringent condition, no? We'll explore it in Exercise 6.75 below.

Let's explain why cofunctors between state categories are just product projections. A product projection $A \times B \to A$ always has another factor (B); if every cofunctor between state categories is a product projection, what is the other factor†? It turns out that the other factor will be:

$$F \coloneqq \{f \colon T \to S \mid t \in T,\ \text{get}(f(t)) = t \text{ and } \text{put}(f(t), t) = f(t)\}.$$

In other words we will see that if (get, put) is a comonoid homomorphism then there is a bijection $S \cong T \times F$ and that get: $S \to T$ is one of the projections. We will see that the converse is true in Exercise 6.74

So assume (get, put): $Sy^S \nrightarrow Ty^T$ is a comonoid homomorphism, in particular that it satisfies put-get, get-put, and put-put. We obtain a function $\pi \colon S \to T \times F$ given by

$$s \mapsto \big(\text{get}(s), t \mapsto \text{put}(s,t)\big)$$

and it is well-defined since for all $s \in S$ and $t, t' \in T$ we have $\text{get}(\text{put}(s,t)) = t$ by put-get and $\text{put}(\text{put}(s,t), t') = \text{put}(s, t')$ by put-put. We also obtain a function $\pi' \colon T \times F \to S$ given by

$$(t, f) \mapsto f(t).$$

The two functions π, π' are mutually inverse: the roundtrip on S is identity because $\text{put}(s, \text{get}(s)) = s$ by get-put; the roundtrip on $T \times F$ is identity because $\text{get}(f(t)) = t$ and $\text{put}(f(t), t) = f(t)$ by assumption on $f \in F$.

Exercise 6.74 (Solution here). Let S, T, F be sets and suppose given an isomorphism $\alpha \colon S \to T \times F$.

1. Show that there exists a very well behaved lens get: $S \to T$ and put: $S \times T \to S$.
2. Show that there exists a cofunctor between the state category on S and the state category on T.
3. Show that there exists a comonoid homomorphism $Sy^S \to Ty^T$ between the state comonoids. ◊

Exercise 6.75 (Solution here).

1. Suppose $|S| = 3$. How many cofunctors are there $Sy^S \to Sy^S$?
2. Suppose $|S| = 4$ and $|T| = 2$. How many cofunctors are there $Sy^S \nrightarrow Ty^T$? ◊

Exercise 6.76 (Solution here). Let S, T be sets and get: $S \to T$ and put: $S \times T \to S$ the parts of a very well behaved lens, i.e. a cofunctor $Sy^S \to Ty^T$ between state categories. Is it possible that put: $S \times T \to S$ is itself a product projection, i.e. sends $(s,t) \mapsto s$? ◊

When we get to cofree comonoids, we'll obtain a whole new class of cofunctors that are interesting to consider. But for now, we move on to more theory.

6.6 Products in $\mathbf{Cat}^\sharp$

Products in $\mathbf{Cat}^\sharp$ are fascinating. Given categories $\mathcal{C}$ and $\mathcal{D}$, the set of objects in their $\mathbf{Cat}$-product is given by the product of their sets of objects, but this is not the case in $\mathbf{Cat}^\sharp$. So what is an object in $\mathcal{C} \times^\sharp \mathcal{D}$ (the usual categorical product, but taken in $\mathbf{Cat}^\sharp$)?

An object in $\mathcal{C} \times^\sharp \mathcal{D}$ is, roughly speaking, a tree for which each node is a pair of objects: some $i \in \mathcal{C}$ and some $j \in \mathcal{D}$. The edges leading out of node (i, j) are then all the morphisms emanating from i and all the morphisms emanating from j. The tree must respect identities, codomains, and composites in $\mathcal{C}$ and $\mathcal{D}$, as we'll explain. But before we do, we define the following category, which will be crucial to our understanding of products in $\mathbf{Cat}^\sharp$.

Definition 6.77 (Free monoidal categories on monoids and comonoids)**.** Given a (small) set I, define Δ_I to be the free monoidal category generated by $|I|$ distinct monoids. Dually, Δ_I^{op} is the free monoidal category generated by $|I|$ distinct comonoids.

Essentially, what this definition says is that Δ_I^{op} is a monoidal category with $|I|$ distinct comonoids, each with its own counit and comultiplication, as well as all the objects and morphisms that can be obtained from these comonoids via composition and taking the monoidal product. These objects and morphisms are then subject to no relations beyond those implied by the standard comonoid axioms.

In particular, we can identify the objects of Δ_I^{op} with the elements of the free monoid $\text{List}(I)$, where the $|I|$ comonoids are the singleton lists $[i]$ for each $i \in I$. The monoidal product of Δ_I^{op} can then be interpreted as list concatenation, which—in a suggestive overloading of notation—we will denote by $\triangleleft$. The monoidal unit must then be the empty list $[]$. We could give analogous notation for Δ_I.

The counits $[i] \to []$ and comultiplications $[i] \to [i, i]$ for each $i \in I$ generate all the morphisms of Δ_I^{op} via composition and taking the monoidal product, while satisfying associativity and left and right unit laws. For instance, with $I \coloneqq \mathtt{3}$, there are two distinct morphisms $[2, 3, 1, 3] \to [2, 2, 3]$ in Δ_I^{op}. One of them is given by

$$([2] \to [2, 2]) \triangleleft \text{id}_{[3]} \triangleleft ([1] \to []) \triangleleft ([3] \to []),$$

and the other is given by

$$([2] \to [2, 2]) \triangleleft ([3] \to []) \triangleleft ([1] \to []) \triangleleft \text{id}_{[3]}.$$

Another way to state Definition 6.77 would be to say that Δ_I^{op} is initial among monoidal categories equipped with $|I|$ comonoids. That is, given any monoidal category $(\mathcal{C}, y, \triangleleft)$ with a comonoid $(c_i, \epsilon_i, \delta_i)$ for each $i \in I$, there is a unique monoidal functor $\Delta_I^{\text{op}} \to \mathcal{C}$ that sends each $[i]$ to c_i, each $[i] \to []$ to ϵ_i, and each $[i] \to [i, i]$ to δ_i. Dually, Δ_I is initial among monoidal categories equipped with $|I|$ monoids.

Example 6.78 (The augmented simplex category). If we take $I := \mathsf{1}$, then $\Delta := \Delta_I$ is what is commonly known as the *augmented simplex category* or the *algebraist's simplex category*. It is the free monoidal category generated by the monoid $[1]$ with unit $[] \to [1]$ and multiplication $[1,1] \to [1]$).

If we identify each n-element list $[1,\ldots,1]$ with the finite set n, interpreted as an ordinal, we can see that Δ is in fact the category of all finite ordinals (i.e. $\mathsf{0},\mathsf{1},\mathsf{2},\ldots$) and the order-preserving maps between them. In [**maclane**], Mac Lane verifies that Δ is initial among monoidal categories equipped with a monoid.

Armed with the free monoidal category Δ_I^{op}, we are now ready to state how products can be constructed in $\mathbf{Cat}^{\sharp}$.

Proposition 6.79. The category $\mathbf{Cat}^{\sharp}$ has all small products.

In particular, for a small set I and a category $\mathcal{C}_i \in \mathbf{Cat}^{\sharp}$ corresponding to a comonoid $\mathfrak{c}_i \in \mathbf{Poly}$ for each $i \in I$, the product of the $\mathcal{C}_i$'s is given by the limit of the canonical monoidal functor $C\colon \Delta_I^{\mathrm{op}} \to \mathbf{Poly}$ sending each $[i]$ to $\mathfrak{c}_i$.

Before we give the proof of the proposition above, let us examine what it says concretely in the case of binary products.

Example 6.80 (Binary products in $\mathbf{Cat}^{\sharp}$). For each $i \in \mathsf{2}$, let $\mathcal{C}_i$ be the category corresponding to the comonoid $(\mathfrak{c}_i, \epsilon_i, \delta_i)$ in $\mathbf{Poly}$. Then we can define $C\colon \Delta_{\mathsf{2}}^{\mathrm{op}} \to \mathbf{Poly}$ to be the canonical monoidal functor sending each $[i]$ to $\mathfrak{c}_i$. Proposition 6.79 asserts that $\lim C$ is the product of $\mathcal{C}_1$ and $\mathcal{C}_2$ in $\mathbf{Cat}^{\sharp}$. But what kind of a polynomial is $\lim C$, and what is its comonoid structure?

Well, for every list $\ell \in \mathrm{Ob}(\Delta_{\mathsf{2}}^{\mathrm{op}}) = \mathrm{List}(\mathsf{2})$, the polynomial $\lim C$ should have a $C\ell$-component, i.e. a projection $\pi_\ell\colon \lim C \to C\ell$. For example, when $\ell := [1,2,2,1,1,1]$, we have $C\ell = \mathfrak{c}_1 \triangleleft \mathfrak{c}_2 \triangleleft \mathfrak{c}_2 \triangleleft \mathfrak{c}_1 \triangleleft \mathfrak{c}_1 \triangleleft \mathfrak{c}_1$, giving us a projection $\lim C \to \mathfrak{c}_1 \triangleleft \mathfrak{c}_2 \triangleleft \mathfrak{c}_2 \triangleleft \mathfrak{c}_1 \triangleleft \mathfrak{c}_1 \triangleleft \mathfrak{c}_1$. These projections must commute with the morphisms in the image of C, which are precisely the morphisms generated by the ϵ_i's and the δ_i's via composition and taking the monoidal product. For instance, the diagram

$$\begin{tikzcd}
\lim C \arrow[r, "\pi_{[1,2,1]}"] \arrow[dr, "\pi_{[1,2,2,1,1,1]}"'] & \mathfrak{c}_1 \triangleleft \mathfrak{c}_2 \triangleleft \mathfrak{c}_1 \arrow[d, "\mathfrak{c}_1 \triangleleft \delta_2 \triangleleft (\delta_1 \fatsemi (\mathfrak{c}_1 \triangleleft \delta_1))"] \\
& \mathfrak{c}_1 \triangleleft \mathfrak{c}_2 \triangleleft \mathfrak{c}_2 \triangleleft \mathfrak{c}_1 \triangleleft \mathfrak{c}_1 \triangleleft \mathfrak{c}_1
\end{tikzcd} \tag{6.81}$$

commutes. In fact, notice that for all $\ell \in \mathrm{List}(\mathsf{2})$, there exists a unique alternating list ℓ' of 1's and 2's with no repetitions (such as $[1,2,1]$, $[2,1,2,1]$, $[]$, or $[2]$) for which there is a unique morphism $d_{\ell,\ell'}\colon \ell' \to \ell$ generated by comultiplications $[i] \to [i,i]$ (here uniqueness is guaranteed by associativity). So we can generalize (6.81) to say that

$$\begin{tikzcd}
\lim C \arrow[r, "\pi_{\ell'}"] \arrow[dr, "\pi_\ell"'] & C\ell' \arrow[d, "Cd_{\ell,\ell'}"] \\
& C\ell
\end{tikzcd} \tag{6.82}$$

commutes.

Hence the family of projections π_ℓ for all $\ell \in \mathrm{List}(2)$ is completely characterized by just the projections $\pi_{\ell'}$ for which ℓ' contains no repetitions. Let us focus, then, on only those projections. Together, they form the commutative diagram

$$\begin{tikzcd}
y & \mathfrak{c}_1 \arrow[l, "\epsilon_1"'] & \mathfrak{c}_1 \triangleleft \mathfrak{c}_2 \arrow[l, "\mathfrak{c}_1 \triangleleft \epsilon_2"'] & \cdots \arrow[l] \\
\lim C \arrow[u, "\pi_{[]}"] \arrow[ur, "\pi_{[1]}"] \arrow[urr, "\pi_{[1,2]}"'] \arrow[d, "\pi_{[]}"'] \arrow[dr, "\pi_{[2]}"'] \arrow[drr, "\pi_{[2,1]}"] & & & \\
y & \mathfrak{c}_2 \arrow[l, "\epsilon_2"] & \mathfrak{c}_2 \triangleleft \mathfrak{c}_1 \arrow[l, "\mathfrak{c}_2 \triangleleft \epsilon_1"] & \cdots . \arrow[l]
\end{tikzcd} \tag{6.83}$$

It follows that there are projections from $\lim C$ to both the limit of the top row of (6.83) and the limit of the bottom row of (6.83). In particular, the limit of the top row of (6.83) can be thought of intuitively as the "infinite" monoidal product $\mathfrak{c}_1 \triangleleft \mathfrak{c}_2 \triangleleft \mathfrak{c}_1 \triangleleft \cdots$, corresponding to the polynomial whose positions are all of the infinite structures that can be constructed by following these instructions:

- 1.1. choose an object $c_1 \in \mathcal{C}_1$;
- 1.2. for each morphism f_1 in $\mathcal{C}_1$ with domain c_1:
  - 2.1. choose an object $d_2 \in \mathcal{C}_2$;
  - 2.2. for each morphism g_2 in $\mathcal{C}_2$ with domain d_2:
    - 3.1. choose an object $c_3 \in \mathcal{C}_1$;
    - 3.2. for each morphism f_3 in $\mathcal{C}_1$ with domain c_3:
      - $\cdots$,

Then the directions at each such position are the sequences of morphisms formed by starting from the top of the instructions above and taking the morphisms mentioned in steps $1.2, 2.2, \ldots, n.2$, for some finite n, to obtain sequences such as $()$, (f_1), (f_1, g_2), and (f_1, g_2, f_3). The limit of the bottom row of (6.83) can be characterized in the same way, but with the roles of $\mathcal{C}_1$ and $\mathcal{C}_2$ swapped. From these structures, we can read off the behavior of each projection $\pi_{\ell'}$ from $\lim C$; for instance, the projection $\pi_{[1,2]}$ sends each position of $\lim C$ to the position of $\mathfrak{c}_1 \triangleleft \mathfrak{c}_2$ specified by following just the first three steps of the above instructions.

So every position of $\lim C$ yields a pair of these structures. We'll call the structure obtained by following the instructions above the *left* structure, and the structure obtained by following the instructions above, but with $\mathcal{C}_1$ and $\mathcal{C}_2$ swapped, the *right* structure. But not every such pair of structures corresponds to a position of $\lim C$; they must satisfy additional conditions, given by morphisms in the image of C that are not depicted in (6.83). For instance, the fact that the diagram

$$\begin{tikzcd}
\lim C \arrow[r, "\pi_{[1,2]}"] \arrow[dr, "\pi_{[2]}"'] & \mathfrak{c}_1 \triangleleft \mathfrak{c}_2 \arrow[d, "\epsilon_{1 \triangleleft \mathfrak{c}_2}"] \\
 & \mathfrak{c}_2
\end{tikzcd} \tag{6.84}$$

commutes implies that, when following the above instructions, the object d_2 chosen for the identity morphism on c_1 in the left structure must also be the first object chosen when constructing the right structure. Similar diagrams render all such objects chosen for identity morphisms when constructing these structures redundant. So, in the end, we have a one-to-one correspondence between positions of $\lim C$ and pairs of structures that can be constructed by following these instructions:

1.1. choose an object $c_1 \in \mathcal{C}_1$;
1.2. for each nonidentity morphism f_1 in $\mathcal{C}_1$ with domain c_1:
  2.1. choose an object $d_2 \in \mathcal{C}_2$;
  2.2. for each nonidentity morphism g_2 in $\mathcal{C}_2$ with domain d_2:
    3.1. choose an object $c_3 \in \mathcal{C}_1$;
    3.2. for each nonidentity morphism f_3 in $\mathcal{C}_1$ with domain c_3:
      . . .

for the left structure, and

1.1. choose an object $d_1 \in \mathcal{C}_2$;
1.2. for each nonidentity morphism g_1 in $\mathcal{C}_2$ with domain d_1:
  2.1. choose an object $c_2 \in \mathcal{C}_1$;
  2.2. for each nonidentity morphism f_2 in $\mathcal{C}_1$ with domain c_2:
    3.1. choose an object $d_3 \in \mathcal{C}_2$;
    3.2. for each nonidentity morphism g_3 in $\mathcal{C}_2$ with domain d_3:
      . . .

for the right structure. These pairs are then the objects of $\mathcal{C}_1 \times^\sharp \mathcal{C}_2$. The morphisms from each object are the sequences of morphisms formed by starting from the top of either set of instructions above and taking the morphisms mentioned in steps 1.2, 2.2, . . . , n.2, for some finite n, to obtain sequences such as $()$, (f_1), (g_1), (f_1, g_2), (g_1, f_2), (f_1, g_2, f_3) and (g_1, f_2, g_3). In particular, the identity morphism is $()$.

We illustrate how to determine the codomain of each nonidentity morphism using an example. To find the codomain of the morphism (g_1, f_2), obtained from steps 1.2 and 2.2 in the right structure, we need to determine the left and right structures of the codomain. Its right structure is easy to describe: it is simply the structure obtained by following the remaining instructions, starting from step 3.1, nested under step 2.2 for f_2; the recursive nature of these instructions ensures that this is, in fact, a valid right structure. As for the left structure of the codomain, we can construct it as follows:

1.1. choose the object $\operatorname{cod}(f_2) \in \mathcal{C}_1$;

1.2. for each nonidentity morphism f_2' in $\mathcal{C}_1$ with domain $\operatorname{cod}(f_2)$:

  2.1. if $f_2 \fatsemi f_2'$ is not the identity, then follow the steps nested under step 2.2 for $f_2 \fatsemi f_2'$ in the instructions above for constructing the original right structure;

  2.2. otherwise,

$\operatorname{cod}(f_2)$ in $\mathcal{C}_1$, then, for each nonidentity morphism f_2' in $\mathcal{C}_1$

Example 6.85 (Products of discrete categories are products of sets). Given sets S and T, consider the corresponding discrete categories Sy and Ty. Then by Example 6.80, the product $Sy \times^\sharp Ty$ is the discrete category $(S \times T)y$.

Example 6.86 (Products of one-object categories are coproducts of monoids). Given monoids M and N, consider the corresponding one-object categories y^M and y^N. Then by Example 6.80, the product $y^M \times^\sharp y^N$ is the one-object category y^{M*N}, where $M * N$ is the free product (i.e. the coproduct) of the monoids M and N.

Example 6.87 (Unexpectedly-many objects). Let $\mathcal{C} := \boxed{\overset{A}{\circ} \xrightarrow{f} \overset{B}{\circ}}$ be the walking arrow category. By Example 6.80, the product $\mathcal{C} \times^\sharp \mathcal{C}$ has infinitely many objects. Namely, it has an object for every pair of sequences (s_1, s_2), finite or infinite, that one can write in the alphabet $\{A_1, A_2, B_1, B_2\}$ subject to the following conditions:

- the sequence s_1 starts with either A_1 or B_1;
- the sequence s_2 starts with either A_2 or B_2; and
- in either sequence:
  - after A_1 comes either A_2 or B_2;
  - after A_2 comes either B_1 or A_1; and
  - after B_1 or B_2, the sequence stops.

For example, here is an object in $\boxed{\overset{A}{\circ} \xrightarrow{f} \overset{B}{\circ}} \times \boxed{\overset{A}{\circ} \xrightarrow{f} \overset{B}{\circ}}$:

$$((A_1, A_2, A_1, A_2, \ldots), (A_2, A_1, A_2, A_1, B_2))$$

Example 6.88. Let I be a set and Iy the associated discrete category, and let $(M, e, *)$ be a monoid and y^M the associated one-object category. Then by Example 6.80, the product

$Iy \times^\sharp y^M$ is the category $I^M y^M$. Its objects are functions $i \colon M \to I$, while its morphisms have the form $i \xrightarrow{m} (m' \mapsto i(m * m'))$ for each $i \colon M \to I$ and $m \in M$.

Given $i \colon M \to I$, its identity morphism is the morphism $i \xrightarrow{e} i$ corresponding to $e \in M$; given $m, n \in M$, the composite of the morphisms $i \xrightarrow{m} (m' \mapsto i(m * m')) \xrightarrow{n} (m' \mapsto i(m * n * m'))$ is the morphism $i \xrightarrow{m*n} (m' \mapsto i(m * n * m'))$.

Proof of Proposition 6.79. We first show that $\lim C$ actually is a comonoid in **Poly** by giving its counit and comultiplication. The counit $\epsilon \colon \lim C \to y$ is given by the projection $\pi_{[]} \colon \lim C \to C[]$, since $C[] \cong y$.

To construct the comultiplication, we observe that Δ_I^{op} is connected (every object has map to [] given by a monoidal product of counits), so by Theorem 5.52, $\triangleleft$ preserves Δ_I^{op}-shaped limits. As C is monoidal, it follows that

$$(\lim C) \triangleleft (\lim C) \cong \lim_{\ell \in \Delta_I^{\mathrm{op}}} \lim_{\ell' \in \Delta_I^{\mathrm{op}}} C(\ell \triangleleft \ell').$$

So specifying a map to $(\lim C) \triangleleft (\lim C)$ amounts to specifying a map to $C(\ell \triangleleft \ell')$ for every $\ell, \ell' \in \Delta_I^{\mathrm{op}}$ such that the appropriate diagrams commute. In particular, we can construct the comultiplication $\delta \colon \lim C \to (\lim C) \triangleleft (\lim C)$ by specifying the projection $\pi_{\ell \triangleleft \ell'} \colon \lim C \to C(\ell \triangleleft \ell')$ for each $\ell, \ell' \in \Delta_I^{\mathrm{op}}$. It is routine to verify that the appropriate diagrams commute.

(Verify comonoid laws)

(Give projections; check that these are comonoid morphisms)

(Verify universal property of product) □

6.7 Some math about Cat$^\sharp$

We refer to morphisms between polynomial comonoids as cofunctors, again eliding the difference between comonoids in **Poly** and categories.

Proposition 6.89. The coproduct of polynomial comonoids agrees with the coproduct of categories. In particular, the initial comonoid is 0.

Proof. We refer the claim about 0 to Exercise 6.90.

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be categories and $\mathfrak{c}_1, \mathfrak{c}_2$ their emanation polynomials, i.e. the carriers of the corresponding comonoids $\mathscr{C}_1$ and $\mathscr{C}_2$. We first notice that $\mathfrak{c} := \mathfrak{c}_1 + \mathfrak{c}_2$ is the carrier for the comonoid $\mathscr{C}$ corresponding to the sum category $\mathcal{C} := \mathcal{C}_1 + \mathcal{C}_2$. Indeed, the object-set of the sum is given by the sum of the object-sets

$$\mathrm{Ob}(\mathcal{C}_1 + \mathcal{C}_2) \cong \mathrm{Ob}(\mathcal{C}_1) + \mathrm{Ob}(\mathcal{C}_2),$$

and a morphism in $\mathcal{C}_1 + \mathcal{C}_2$ emanating from any such object is just a morphism in whichever of the categories $\mathcal{C}_1$ or $\mathcal{C}_2$ it is from.

It remains to show that $\mathscr{C}$ is the coproduct of $\mathscr{C}_1$ and $\mathscr{C}_2$ in **Cat**$^\sharp$. Suppose given a comonoid $\mathscr{D}$ and comonoid homomorphisms (cofunctors) $f_1 \colon \mathscr{C}_1 \nrightarrow \mathscr{D}$ and $f_2 \colon \mathscr{C}_2 \nrightarrow \mathscr{D}$.

Then for any object of $\mathscr{C}$ we have an associated object $f(c) \in \mathscr{D}$, given either by $f(c) \coloneqq f_1(c)$ or by $f(c) \coloneqq f_2(c)$ depending on whether $c \in \mathscr{C}_1$ or $c \in \mathscr{C}_2$. For any morphism m emanating from $f(c)$ we have a morphism $f^\sharp(m)$ emanating from c. It is easy to check that the cofunctor laws hold for f. Uniqueness of f given f_1, f_2 is also straightforward. □

Exercise 6.90 (Solution here).

1. Show that 0 is a comonoid.
2. Show that 0 is initial as a comonoid. ◊

Exercise 6.91 (Solution here). If $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ is a category, show there is an induced category structure on the polynomial $2\mathfrak{c}$. ◊

Exercise 6.92 (Solution here). Check that the terminal comonoid is y. ◊

Proposition 6.93 (Porst)**.** The forgetful functor **Comon**(**Poly**) → **Poly** is comonadic.

Proof. The fact that a forgetful functor **Comon**(**Poly**) → **Poly** is comonadic if it has a right adjoint follows from Beck's monadicity theorem via a straightforward generalization of an argument given by Paré in [Par69, pp. 138-9], as pointed out by Porst in [Por19, Fact 3.1]. □

Corollary 6.94. The category $\mathbf{Cat}^\sharp = \mathbf{Comon}(\mathbf{Poly})$ has all small colimits. They are created by the underlying polynomial functor **Comon**(**Poly**) → **Poly**.

Proof. It is well known that a comonadic functor creates all colimits that exist in its codomain [nLa18]. By Theorem 4.42, the category **Poly** has all small colimits. □

Proposition 6.95. Let $\mathbf{Comon}(\mathbf{Poly})_{\text{rep}}$ be the full subcategory of comonoids (c, ϵ, δ) in **Poly** for which the carrier $c = y^M$ is representable. Then there is an isomorphism of categories

$$\mathbf{Comon}(\mathbf{Poly})_{\text{rep}} \cong \mathbf{Mon}^{\text{op}}$$

where **Mon** is the category of monoids.

Proof. Let $\mathcal{C}$ be a category. It has only one object iff its emanation polynomial $\mathfrak{c}$ has only one position, i.e. $\mathfrak{c} \cong y^M$ for some $M \in \mathbf{Set}$, namely where M is the set of morphisms in $\mathcal{C}$. It remains to show that cofunctors between monoids are dual—opposite—to morphisms between monoids.

A cofunctor $f\colon y^M \to y^N$ involves a single function $f^\sharp\colon N \to M$ that must satisfy a law coming from unitality and one coming from composition, as in Definition 6.42. The result can now be checked by hand, or seen formally as follows. Each object in the two diagrams (6.43) is representable by Exercise 5.9. The Yoneda embedding $\mathbf{Set}^{\mathrm{op}} \to \mathbf{Poly}$ is fully faithful, so these two diagrams are equivalent to the unit and composition diagrams for monoid homomorphisms. □

Exercise 6.96 (Solution here). Let $\mathbf{Comon}(\mathbf{Poly})_{\mathrm{lin}}$ be the full subcategory of comonoids (c, ϵ, δ) in **Poly** for which the carrier $c = My$ is linear. Show that there is an isomorphism of categories

$$\mathbf{Comon}(\mathbf{Poly})_{\mathrm{lin}} \cong \mathbf{Set}. \qquad \lozenge$$

Proposition 6.97. The inclusion of linear comonoids into all comonoids has a left adjoint

$$\begin{tikzcd} \mathbf{Comon}(\mathbf{Poly}) \arrow[r, shift left=2, "(\mathfrak{c}\triangleleft 1)y"] \arrow[r, phantom, "\Rightarrow"] & \mathbf{Comon}(\mathbf{Poly})_{\mathrm{lin}} \arrow[l, shift left=2, "Ay"] \end{tikzcd}$$

denoted by where they send a comonoid $(\mathfrak{c}, \epsilon, \delta)$ and a linear comonoid Ay.

Proof. We need to show that for any comonoid $(\mathfrak{c}, \epsilon, \delta)$ and set A, we have a natural isomorphism

$$\mathbf{Cat}^\sharp(\mathfrak{c}, Ay). \cong^? \mathbf{Cat}^\sharp((\mathfrak{c}\triangleleft 1)y, Ay)$$

But every morphism in Ay is an identity, so the result follows from the fact that every cofunctor must pass identities back to identities. □

A cofunctor (map of polynomial comonoids) is called *cartesian* if the underlying map $f\colon \mathfrak{c} \to \mathfrak{d}$ of polynomials is cartesian (i.e. for each position $i \in \mathfrak{c}(1)$, the map $f_i^\sharp\colon \mathfrak{d}[f_1(i)] \to \mathfrak{c}[i]$ is an isomorphism).

Proposition 6.98. Every cofunctor $f\colon \mathcal{C} \nrightarrow \mathscr{D}$ factors as a vertical morphism followed by a cartesian morphism

$$\mathcal{C} \overset{\text{vert}}{\nrightarrow} \mathcal{C}' \overset{\text{cart}}{\nrightarrow} \mathscr{D}.$$

Proof. A cofunctor $\mathcal{C} \nrightarrow \mathscr{D}$ is a map of polynomials $\mathfrak{c} \to \mathfrak{d}$ satisfying some properties, and any map of polynomials $f\colon \mathfrak{c} \to \mathfrak{d}$ can be factored as a vertical morphism followed by a cartesian morphism

$$\mathfrak{c} \xrightarrow{g} \mathfrak{c}' \xrightarrow{h} \mathfrak{d}.$$

For simplicity, assume $g_1\colon \mathfrak{c}(1) \to \mathfrak{c}'(1)$ is identity (rather than merely isomorphism) on positions and similarly that for each $i \in \mathfrak{c}$ the map $h_i^\sharp\colon \mathfrak{c}'[i] \to \mathfrak{d}[h_1(i)]$ is identity (rather than merely isomorphism) on directions.

It suffices to show that the intermediate object $\mathfrak{c}'$ can be endowed with the structure of a category such that g and h are cofunctors. Given an object $i \in \mathfrak{c}'(1)$, assign its identity to be the identity on $h_1(i) = f(i)$; then both g and h preserve identities because f does. Given an emanating morphism $m \in \mathfrak{c}'[i] = \mathfrak{d}[f(i)]$, assign its codomain to be $\operatorname{cod}(m) := \operatorname{cod}(f_i^\sharp(m))$, and given an emanating morphism $m' \in \mathfrak{c}'[\operatorname{cod}(m)]$, assign the composite $m \mathbin{⨟} m'$ in $\mathfrak{c}'$ to be $m \mathbin{⨟} m'$ in $\mathfrak{d}$. In Exercise 6.99 we will check that with these definitions, $\mathfrak{c}'$ is a category and both g and h are cofunctors. □

Exercise 6.99 (Solution here). We will complete the proof of Proposition 6.98, using the same notation.

1. Show that composition is associative and unital in $\mathfrak{c}'$.
2. Show that g preserves codomains.
3. Show that g preserves compositions.
4. Show that h preserves codomains.
5. Show that h preserves compositions. ◊

Proposition 6.100. The wide subcategory of cartesian maps in $\mathbf{Cat}^\sharp$ is isomorphic to the category of wide subcategory of discrete opfibrations in **Cat**.

Proof. Suppose that $\mathcal{C}$ and $\mathcal{D}$ are categories. Both a functor and a cofunctor between them involve a map on objects, say $f\colon \operatorname{Ob}(\mathcal{C}) \to \operatorname{Ob}(\mathcal{D})$. For any object $c \in \operatorname{Ob}(\mathcal{C})$, a functor gives a function, say $f_\sharp\colon \mathcal{C}[c] \to \mathcal{D}[f(c)]$ whereas a cofunctor gives a function $f^\sharp\colon \mathcal{D}[f(c)] \to \mathcal{C}[c]$. The cofunctor is cartesian iff $f^\sharp$ is an iso, and the functor is a discrete opfibration iff $f_\sharp$ is an iso. We thus transform our functor into a cofunctor (or vice versa) by taking the inverse function on morphisms. It is easy to check that this inverse appropriately preserves identities, codomains, and compositions. □

Proposition 6.101. The wide subcategory of vertical maps in $\mathbf{Cat}^\sharp$ is isomorphic to the opposite of the wide subcategory bijective-on-objects maps in **Cat**:

$$\mathbf{Cat}^\sharp_{\text{vert}} \cong (\mathbf{Cat}_{\text{boo}})^{\text{op}}.$$

Proof. Let $\mathcal{C}$ and $\mathcal{D}$ be categories. Given a vertical cofunctor $F\colon \mathcal{C} \nrightarrow \mathcal{D}$, we have a bijection $F_1\colon \operatorname{Ob}(\mathcal{C}) \to \operatorname{Ob}(\mathcal{D})$; let G_1 be its inverse. We define a functor $G\colon \mathcal{D} \to \mathcal{C}$ on objects by G_1 and, for any $f\colon d \to d'$ in $\mathcal{D}$ we define $G(f) := F^\sharp_{G_1(d)}$. It has the correct codomain: $\operatorname{cod}(G(f)) = G_1(F_1(\operatorname{cod}(G(f))) = G_1(\operatorname{cod} f)$. And it sends identities and compositions to identities and compositions by the laws of cofunctors.

The construction of a vertical cofunctor from a bijective-on-objects functor is analogous, and it is easy to check that the two constructions are inverses. □

Exercise 6.102 (Solution here). Let S be a set and consider the state category $\mathcal{S} := (Sy^S, \epsilon, \delta)$. Use Proposition 6.101 to show that categories $\mathcal{C}$ equipped with a vertical cofunctor $\mathcal{S} \to \mathcal{C}$ can be identified with categories whose set of objects is S. ◊

Exercise 6.103 (Solution here). Consider the categories $\mathcal{C} := \boxed{\bullet \rightrightarrows \bullet}$ and $\mathcal{D} := \boxed{\bullet \to \bullet}$. There is a unique bijective-on-objects (boo) functor $F \colon \mathcal{C} \to \mathcal{D}$ and two boo functors $G_1, G_2 \colon \mathcal{D} \to \mathcal{C}$.

1. Write down the morphism $\mathfrak{d} \to \mathfrak{c}$ of emanation polynomials underlying F.
2. Write down the morphism $\mathfrak{c} \to \mathfrak{d}$ of emanation polynomials underlying either G_1 or G_2. ◊

6.7.1 Dirichlet monoidal product on Cat$^\sharp$

The usual product of categories gives a monoidal operation on comonoids too, even though it is not a product in **Cat**$^\sharp$. The carrier polynomial of the product is the $\otimes$-product of the carrier polynomials.

Proposition 6.104. The Dirichlet monoidal product $(y, \otimes)$ on **Poly** extends to a monoidal structure $(y, \otimes)$ on **Cat**$^\sharp$, such that the functor $U \colon \mathbf{Cat}^\sharp \to \mathbf{Poly}$ is strong monoidal with respect to $\otimes$. The Dirichlet product of two categories is their product in **Cat**.

Proof. Let $\mathcal{C}, \mathcal{D} \in \mathbf{Cat}^\sharp$ be categories with emanation polynomials $\mathfrak{c}, \mathfrak{d} \in \mathbf{Poly}$. The emanation polynomial of $\mathcal{C} \otimes \mathcal{D}$ is defined to be $\mathfrak{c} \otimes \mathfrak{d}$. A position in it is a pair (c, d) of objects, one from $\mathcal{C}$ and one from $\mathcal{D}$; a direction there is a pair (f, g) of a morphism emanating from c and one emanating from d.

We define $\epsilon_{\mathcal{C}\otimes\mathcal{D}} \colon \mathfrak{c} \otimes \mathfrak{d} \to y$ as

$$\mathfrak{c} \otimes \mathfrak{d} \xrightarrow{\epsilon_{\mathfrak{C}} \otimes \epsilon_{\mathfrak{D}}} y \otimes y \cong y.$$

This says that the identity at (c, d) is the pair of identities.

We define $\delta_{\mathcal{C}\otimes\mathcal{D}} \colon (\mathfrak{c} \otimes \mathfrak{d}) \to (\mathfrak{c} \otimes \mathfrak{d}) \mathbin{\triangleleft} (\mathfrak{c} \otimes \mathfrak{d})$ using the duoidal property:

$$\mathfrak{c} \otimes \mathfrak{d} \xrightarrow{\delta_{\mathfrak{c}} \otimes \delta_{\mathfrak{d}}} (\mathfrak{c} \mathbin{\triangleleft} \mathfrak{c}) \otimes (\mathfrak{d} \mathbin{\triangleleft} \mathfrak{d}) \to (\mathfrak{c} \otimes \mathfrak{d}) \mathbin{\triangleleft} (\mathfrak{c} \otimes \mathfrak{d}).$$

One can check that this says that codomains and composition are defined coordinatewise, and that $(\mathfrak{c} \otimes \mathfrak{d}, \epsilon_{\mathfrak{c}\otimes\mathfrak{d}}, \delta_{\mathfrak{c}\otimes\mathfrak{d}})$ forms a comonoid. One can also check that this is functorial in $\mathcal{C}, \mathcal{D} \in \mathbf{Cat}^\sharp$. See Exercise 6.105. □

Exercise 6.105 (Solution here). We complete the proof of Proposition 6.104.

1. Show that $(\mathfrak{c} \otimes \mathfrak{d}, \epsilon_{\mathfrak{c}\otimes\mathfrak{d}}, \delta_{\mathfrak{c}\otimes\mathfrak{d}})$, as described in Proposition 6.104, forms a comonoid.

2. Check that the construction $(\mathcal{C},\mathcal{D}) \mapsto \mathcal{C} \otimes \mathcal{D}$ is functorial in $\mathcal{C},\mathcal{D} \in \mathbf{Cat}^{\sharp}$. ◊

6.8 Exercise solutions

Solution to Exercise 6.21.

We seek the number of admissible sections of the category $\boxed{\bullet \to \bullet}$. There are 2 choices of morphisms emanating from the object on the left, and 1 choice of morphism emanating from the object on the right, for a total of $2 \cdot 1 = 2$ admissible sections.

Solution to Exercise 6.28.

1. The category $\boxed{A \xrightarrow{f} B}$ has 2 morphisms out of A and 1 morphism out of B, so its emanation polynomial is $y^2 + y$.
2. The category $\boxed{B \xleftarrow{f} A \xrightarrow{g} C}$ has 3 morphisms out of A and 1 morphism out of each of B and C, so its emanation polynomial is $y^3 + 2y$.
3. The empty category has no objects, so its emanation polynomial is the empty sum 0.
4. The monoid $(\mathbb{N}, 0, +)$ has 1 object, and its morphisms form the set $\mathbb{N}$, so its emanation polynomial is $y^{\mathbb{N}}$.
5. The monoid $(M, e, *)$ has 1 object, and its morphisms form the set M, so its emanation polynomial is y^M.
6. The poset $(\mathbb{N}, \leq)$ has $\mathbb{N}$ as its set of objects, and there is exactly one morphism from every $n \in \mathbb{N}$ to each element of $\{n' \in \mathbb{N} \mid n \leq n'\} \cong \mathbb{N}$ (and no other morphisms from n), so the emanation polynomial of the poset is $\mathbb{N}y^{\mathbb{N}}$.
7. The poset $(\mathbb{N}, \geq)$ has $\mathbb{N}$ as its set of objects, and there is exactly one morphism from every $n \in \mathbb{N}$ to each element of $\{0, 1, \ldots, n\} \cong \mathsf{n+1}$ (and no other morphisms from n), so the emanation polynomial of the poset is $\sum_{n \in \mathbb{N}} y^{\mathsf{n+1}} \cong y^1 + y^2 + y^3 + \cdots$.

Solution to Exercise 6.34.

Here $(M, e, *)$ is a monoid, S is a set, $\alpha \colon M \times S \to S$ is an action, and $\mathcal{A}$ is the associated category with emanation polynomial Sy^M; in particular, for each $s \in S$ and $m \in M$, there is a morphism $s \xrightarrow{m} \alpha(m, s)$.

1. We wish to identify the identity morphism for each object $s \in \mathcal{A}$. This should be a morphism whose domain and codomain are s. By the laws of monoid actions, $\alpha(m, s)$ is guaranteed to be s if $m = e$. Hence, it only makes sense for the identity morphism of s to be the morphism $s \xrightarrow{e} s$.
2. Given a morphism $s \xrightarrow{m} \alpha(m, s)$, we wish to determine its composite with another morphism $\alpha(m, s) \xrightarrow{n} \alpha(n, \alpha(m, s))$. By the laws of monoid actions, we have that $\alpha(n, \alpha(m, s)) = \alpha(n * m, s)$, so it makes sense for the composite $s \xrightarrow{m} \alpha(m, s) \xrightarrow{n} \alpha(n, \alpha(m, s))$ to be the morphism $s \xrightarrow{n*m} \alpha(n * m, s)$.

Solution to Exercise 6.37.

The linear polynomial Ay corresponds to a category whose objects form the set A and whose only morphisms are identities: in other words, it is the discrete category on A.

Solution to Exercise 6.39.

1. If every object in $\mathcal{C}$ is linear, then the only morphisms in $\mathcal{C}$ are the identity morphisms, so $\mathcal{C}$ must be a discrete category.
2. It is not possible for an object in $\mathcal{C}$ to have degree 0, as every object must have at least an identity morphism emanating from it.
3. Some possible examples of categories with objects of degree $\mathbb{N}$ are the monoid $(\mathbb{N}, 0, +)$ (see Exercise 6.28 #4), the poset $(\mathbb{N}, \leq)$ (see Exercise 6.28 #6), and the state category on $\mathbb{N}$ (see Example 6.36).

4. There are 3 categories with just one linear and one quadratic object. They can be distinguished by the behavior of the single non-identity morphism. Either its domain and its codomain are distinct, in which case we have the walking arrow category; or its domain and its codomain are the same, in which case it can be composed with itself to obtain either itself or the identity, yielding two more categories for a total of three.
5. Yes: since categories correspond to comonoids, there are as many categories with one linear and one quadratic object as there are comonoid structures on $y^2 + y$.

Solution to Exercise 6.40.

1. Take the discrete category on n and adjoin a unique initial object A, so that the only non-identity morphisms are the morphisms from A to each of the other objects exactly once. Then this category has the emanation polynomial $y^{\mathsf{n}+1} + \mathsf{n}y$.
2. This category could be thought of as "star-shaped," with the initial object in the center with morphisms leading out to the other n objects as spokes.

Chapter 7

Cofree polynomial comonoids

7.1 Introduction: Cofree comonoids and discrete dynamical systems

We can now return to dynamical systems. Recall that if $p \in \mathbf{Poly}$ is a polynomial, then a dynamical system with interface p consists of a set S and a map of polynomials $f\colon Sy^S \to p$. We think of positions in p kind of like outputs—others can observe your position—but also as determining the set of inputs—directions—that could be input next.

The way this map f leads to something that seems to "go on repeatedly" is that $\mathfrak{s} := Sy^S$ is a comonoid, so we have maps $\mathfrak{s} \xrightarrow{\delta} \mathfrak{s}^{\triangleleft n} \xrightarrow{f^{\triangleleft n}} p^{\triangleleft n}$ for all n. This says that given any initial position in S, we automatically get a position in p, and for every direction there, another position in p, and for every direction there, another position in p, and so on n times. This is the dynamics.

So the above all works because we have a polynomial map $Sy^S \to p$, where Sy^S is the underlying polynomial of a polynomial comonad. Since "underlying polynomial" is a functor $U\colon \mathbf{Cat}^\sharp \to \mathbf{Poly}$, a seasoned category theorist might be tempted to ask the following: is there an adjunction

$$\mathbf{Poly}(U\mathscr{C}, p) \cong \mathbf{Cat}^\sharp(\mathscr{C}, \mathscr{T}_p),$$

for some functor $\mathscr{T}$: $\mathbf{Poly} \to \mathbf{Cat}^\sharp$? In fact, there is; we refer to $\mathscr{T}_p$ as the *cofree comonoid* on p, or more descriptively, the *category of p-trees.*

Cofree comonoids in **Poly** are beautiful objects, both in their visualizable structure as a category and in the metaphors we can make about them. They allow us to replace the interface of a dynamical system with a category and get access to a rich theory that exists there.

Theorem 7.1 (Cofree comonoid)**.** The forgetful functor **Comon**(**Poly**) $\to$ **Poly** has a right adjoint

$$\begin{tikzcd} \mathbf{Cat}^\sharp \arrow[r, shift left=2, "\mathfrak{c}"] \arrow[r, phantom, "\Rightarrow"] & \mathbf{Poly} \arrow[l, shift left=2, "\mathscr{T}_p"] \end{tikzcd}$$

where the functors have been named by where they send $(\mathfrak{c}, \epsilon, \delta) \in \mathbf{Cat}^\sharp$ and $p \in \mathbf{Poly}$ respectively.

This will be proved as **??**.

7.2 Cofree comonoids as trees

Definition 7.2 (Cofree comonoid)**.** Let $p \in \mathbf{Poly}$ be a polynomial. The comonoid $\mathscr{T}_p = (\mathsf{tree}_p, \text{root}, \text{focus})$ as in Theorem 7.1 is called the *cofree comonoid on p* or informally the category of *(possibly infinite) p-trees*.

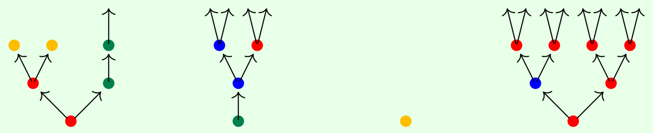
An object $t \in \mathsf{tree}_p(1)$ is a called a *(possibly infinite) tree in p*. Given such an object t in the category, an emanating morphism $n \in \mathsf{tree}_p[t]$ is called a *path from root*.

The terminology of Definition 7.2 is alluding to a specific way we like to imagine the comonoid $\mathscr{T}_p = (\mathsf{tree}_p, \text{root}, \text{focus})$, namely in terms of trees. To every polynomial p, we will associate a new polynomial tree_p whose positions are (possibly infinite) p-trees. To choose such a tree we first choose its root to be some position $i \in p(1)$. Then for every direction $d \in p[i]$ there, we choose another position, and for every direction from each of those we choose another position, and so on indefinitely.

So a position in tree_p is one of these trees. Such a tree may end, namely if none of the top-level positions has any directions, but often it will not end. Given such a tree, say t, a direction $d \in \mathsf{tree}_p[t]$ there is simply a path from the root to some node in the tree.

We'll explain the counit root and the comultiplication focus after going through an example.

Example 7.3. Let $p := \{\bullet, \bullet\}y^2 + \bullet y + \bullet$. Here are four trees in p:



(7.4)

They all represent elements of $p^{\triangleleft 3}$, but only the third one—the single yellow dot—would count as an element of tree_p.

Indeed, in Definition 7.2, when we speak of a (possibly infinite) tree in p, we mean at tree for which each node is a position in p with each of its emanating directions filled by another position in p, and so on. Since three of the four trees shown in (7.4) have open leaves—arrows emanating from the top—these trees are not elements of tree_p. However, each of them could be extended to an actual element of tree_p by continually filling in each open leaf with another position of p.

Let's imagine some actual elements of tree_p:

- The binary tree that's "all red all the time."
- The binary tree where odd layers are red and even layers are blue.
- The tree where all the nodes are red, except for the right-most branch, which is always green.[a]
- Any finite tree, where every branch terminates in a yellow dot.
- Completely random: for the root, randomly choose either red, blue, green, or yellow, and at every leaf, loop back to the beginning, i.e. randomly choose either red, blue, green, or yellow, etc.

These are the positions in the polynomial tree_p that underlies the cofree comonoid on p. There are uncountably many positions in $\mathscr{T}_p$, at least for this particular p—in fact, even $\mathscr{T}_{2y}$ has $2^{\mathbb{N}}$-many positions—but only finitely many can be described in any finite language like English. Thus most of the elements of $\mathscr{T}_p$ cannot be described.

[a] Note that branches are actually unordered, so it's technically wrong to think of it as a line of green up the *right side*. Instead, it's just a line of green going up the tree forever.

Exercise 7.5 (Solution here).

1. Interpret each of the five tree examples imagined in Example 7.3 by drawing three or four layers (your choice) of it.

For each of the following polynomials p, describe the set of trees (positions) in tree_p.

2. $p = 1$. (What is the set tree_p of p-trees?)
3. $p = 2$.
4. $p = y$.
5. $p = y^2$.
6. $p = 2y$.
7. $p = y + 1$.
8. $p = By^A$ for some sets $A, B \in \mathbf{Set}$. ◊

Now that we've explained the underlying polynomial tree_p of the cofree comonoid $\mathscr{T}_p = (\mathsf{tree}_p, \text{root}, \text{focus})$, we just need to explain how identities, codomains, and composition work, i.e. we just need to give the counit map $\text{root}\colon \mathsf{tree}_p \to y$ and the comultiplication map $\text{focus}\colon \mathsf{tree}_p \to \mathsf{tree}_p \triangleleft \mathsf{tree}_p$.

Again, the objects in the category $\mathscr{T}_p$ are p-trees, and a morphism emanating from such a tree t is a path from its root r to some node. The map root, applied to t, returns t's root r, or more precisely the path from r to itself. The map focus, applied to t, first needs to give a codomain (tree) to every path from the root to some other node n. It is just the subtree of t whose root is n: the tree of all nodes starting at n. Now, given a path from the root of that tree (namely n) to another node, say n', we need to give a path from r to n'; we take it to be the composite of the path $r \to n$ and the path $n \to n'$.

Exercise 7.6 (Solution here). Let $p := \{\bullet, \bullet\}y^2 + \bullet y + \bullet$ as in Example 7.3.

1. Choose an object $t \in \mathsf{tree}_p$, i.e. a tree in p, and draw a finite approximation of it (say four layers).
2. What is the identity morphism at t?
3. Choose a nonidentity morphism f emanating from t and draw it.
4. What is the codomain of f? Draw a finite approximation of it.
5. Choose a morphism emanating from the codomain of f and draw it.
6. What is the composite of your two morphisms? Draw it on t. ◊

Example 7.7. Let's take $p := 1$. An element in $\mathsf{tree}_p(1)$ is given by choosing an element $i \in p(1)$ and filling each of its direction $p[i]$ with another element of $p(1)$, and so on. But there is only one element of $p(1)$ and it has no directions. So tree_1 has only one position, and the only emanating morphism there is the identity. In other words, $\mathsf{tree}_1 = y$.

Since y has a unique comonoid structure, we've described the cofree comonoid $\mathscr{T}(1)$. It is a single tree consisting of a single node, and the only outgoing morphism is the identity on that node.

Exercise 7.8 (Solution here). Let A be a set.

1. What is tree_A?
2. How is it given the structure of a category? ◊

Example 7.9. Let $p := y$. An element in $\mathsf{tree}_p(1)$ is given by choosing an element $i \in p(1)$ and filling each of its direction $p[i]$ with another element of $p(1)$, and so on. There is only one way to do this, i.e. there is only one such tree, namely $t := \boxed{\bullet \to \bullet \to \bullet \to \cdots}$.

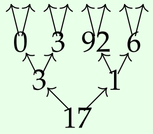
So tree_p has a single position, namely t. That position has an emanating morphism for each path out of the root, so it has $\mathbb{N}$-many emanating morphisms: one for every length. Hence $\mathsf{tree}_y = y^{\mathbb{N}}$.

Of course the codomain of each morphism emanating from t is again t: that's the only object. The composite of two paths, one of length m and one of length n is $m + n$. Hence we see that the category $\mathscr{T}(y)$ is the monoid $(\mathbb{N}, 0, +)$ considered as a category with one object.

Exercise 7.10 (Solution here). Let A be a set.

1. What is tree_{Ay}?
2. How is it given the structure of a category? ◊

Example 7.11. Let $p := \mathbb{N}y^2$. An element of tree_p might start like this:



Any element of tree_p goes on forever: it's an infinite binary tree. At each node it has a choice of some natural number, since $\mathbb{N} = p(1)$ is the position-set of p.

So such trees are the objects of the category $\mathscr{T}_p = (\mathsf{tree}_p, \text{root}, \text{focus})$. A morphism emanating from a tree t is a path from its root to another node, which is an element of $\mathsf{List}(2)$, i.e. a finite list of choices in 2, which you can think of as a finite sequence of left/right choices. The codomain is whatever tree this path ends up on.

So the emanation polynomial of $\mathscr{T}_p$ is

$$\mathsf{tree}_p \cong \mathbb{N}^{\mathsf{List}(2)}y^{\mathsf{List}(2)}$$

with identities given by the empty list. An object $t \in \mathsf{tree}_p(1)$ is a function $t\colon \mathsf{List}(2) \to \mathbb{N}$, a way to put a natural number at every node of the infinite binary tree. An emanating morphism $\ell \in \mathsf{List}(2)$ is just a path from the root to another node, and its codomain is the other node. Formally it is the function $t'\colon \mathsf{List}(2) \to \mathbb{N}$ given by $t'(\ell') := t(\ell : \ell')$, where $\ell : \ell'$ is the concatenation of these lists. Composition of morphisms is also given by concatenation of the corresponding lists.

Exercise 7.12 (Solution here). Let $p := By^A$ for sets $A, B \in \mathbf{Set}$.

1. Describe the objects of the cofree category $\mathscr{T}_p$.
2. For a given such object, describe the emanating morphisms.
3. Describe how to take the codomain of a morphism. ◊

Example 7.13 ($\mathscr{T}_{Ay}$ for linear polynomials). Let $A \in \mathbf{Set}$ be a set. The cofree comonoid $\mathscr{T}_{Ay}$ on the associated linear polynomial has as emanation polynomial $\mathsf{tree}_{Ay} \cong (Ay)^{\mathbb{N}}$. Its objects are A-streams. For each stream $t\colon \mathbb{N} \to A$, an emanating morphism is just an element $n \in \mathbb{N}$. The identity is $0 \in \mathbb{N}$, the codomain of n is the composite function $\mathbb{N} \xrightarrow{+n} \mathbb{N} \xrightarrow{t} A$, and if we denote it by $n\colon t \to (t+n)$ then the composite of morphisms n, n' is $(n+n')\colon t \to (t+n+n')$.

We first saw this category in Example 6.18

Exercise 7.14 (Solution here). Let $p \coloneqq y + 1$.

1. Describe the objects of the cofree category $\mathscr{T}_p$.
2. For a given such object, describe the emanating morphisms.
3. Describe how to take the codomain of a morphism. ◊

Exercise 7.15 (Solution here). Let $p \coloneqq \{a, b, c, \ldots, z, \sqcup\}y + 1$.

1. Describe the objects of the cofree category $\mathscr{T}_p$, and draw one.
2. For a given such object, describe the emanating morphisms.
3. Describe how to take the codomain of a morphism. ◊

Exercise 7.16 (Solution here). Let p be a polynomial, let $Q \coloneqq \{q \in \mathbb{Q} \mid q \geq 0\}$ and consider the monoid y^Q of nonnegative rational numbers under addition. Is it true that any cofunctor $\varphi\colon \mathscr{T}_p \nrightarrow y^{\mathbb{Q}}$ is constant, i.e. that it factors as

$$\mathscr{T}_p \nrightarrow y \nrightarrow y^{\mathbb{Q}}?$$

◊

7.2.1 Decision trees

When you talk about your future, what exactly might you be talking about? In some sense you can make choices that change what will happen to you, but in another sense it's as though for each such choice there is something beyond your control that makes a new situation for you. You're constantly in the position of needing to make a choice, but its results are beyond your control.

This is very much how positions $t \in \mathscr{T}_p$ look. Such a position is a decision tree: at each stage (node), you have an element $i \in p(1)$, which we've been calling a decision. It has $p[i]$-many options, each of which, say $d \in p[i]$ results in a new node $\text{cod}(d)$ of the tree.

So a position t is like a future: it is a current decision, and for every option there, a new decision tree. It's all the decisions you could possibly make, and for each actual choice, it's a new future. A direction at t is just a choice of finite path through the tree: a sequence of choices.

Exercise 7.17 (Solution here). If someone says that they understand a future to be a decision tree $t \in \mathscr{T}_p$, explain in your own words how they're thinking about the term "future." How does it agree or disagree with your own intuition about what a "future" is? ◊

Exercise 7.18 (Solution here). Let G be a finite directed graph, and let $\mathbf{Fr}(G)$ be the associated free category.

1. Construct a cofunctor $\mathbf{Fr}(G) \nrightarrow \mathscr{T}_{1+y+y^2+y^3+\cdots}$.
2. Would you say it associates to each node in G its "future" decision tree? ◊

7.3 Formal construction of $\mathscr{T}_p$

We will sketch how one can formally construct $\mathscr{T}_p$ from p. The first step is called copointing, and it's pretty easy: just multiply p by y. It adds a kind of "default" direction to each position in p.

7.3.1 Copointing

Definition 7.19 (Copointed polynomial)**.** A *copointed polynomial* is a pair (p, ϵ), where $p \in \mathbf{Poly}$ is a polynomial and $\epsilon \colon p \to y$ is a morphism in **Poly**.

A *morphism* of copointed polynomials $f \colon (p, \epsilon) \to (p', \epsilon')$ is a morphism $f \colon p \to p'$ such that $\epsilon = f \mathbin{⨾} \epsilon'$.

Comonoids in **Poly** are triples (p, ϵ, δ), with (p, ϵ) a copointed polynomial, so there are forgetful functors

$$\mathbf{Comon}(\mathbf{Poly}) \to \mathbf{Cpt}(\mathbf{Poly}) \to \mathbf{Poly}.$$

We want to find the right adjoint to the composite—that's the functor $\mathscr{T}_- \colon \mathbf{Poly} \to \mathbf{Comon}(\mathbf{Poly})$—and we will obtain it in two steps.

Proposition 7.20. For any polynomial p, the polynomial py is naturally copointed by the projection to y, and the functor sending $p \mapsto py$ is right adjoint to the forgetful functor

$$\begin{tikzcd} \mathbf{Cpt}(\mathbf{Poly}) \arrow[r, shift left=2, "py"] \arrow[r, phantom, "\Rightarrow"] & \mathbf{Poly} \arrow[l, shift left=2, "q"] \end{tikzcd},$$

where the functors are named by where they send $p \in \mathbf{Poly}$ and $(q, \epsilon) \in \mathbf{Cpt}(\mathbf{Poly})$.

Proof. Clearly the product $py \cong p \times y$ is copointed by the projection map, call it $\pi \colon py \to y$, and the map $p \mapsto py$ is functorial in p. For any copointed polynomial $q \xrightarrow{\epsilon} y$, there is an obvious bijection between morphisms of polynomials $q \to p$ and commutative triangles

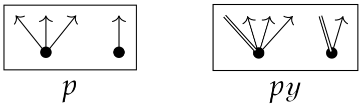
$$\begin{tikzcd} q \arrow[rr] \arrow[dr, "\epsilon"'] & & py \arrow[dl, "\pi"] \\ & y & \end{tikzcd}$$

natural in both q and p. This completes the proof. □

Exercise 7.21 (Solution here). Show that the copointing functor is essentially surjective. That is, every polynomial p equipped with a map $\epsilon\colon p \to y$ is isomorphic to one of the form $p'y$ (equipped with the projection $p'y \to y$). ◊

The reader might not remember any sort of copointing showing up in the tree description of $\mathscr{T}_p = (\mathsf{tree}_p, \text{root}, \text{focus})$. Indeed, it was hidden in the fact that we allowed for trivial paths in the tree (e.g. the path from the root to itself). But we'll get to that.

The copointing $p \mapsto py$ just adds an extra direction to each position; we can denote this extra direction with an =, as we did in Example 6.15. So for example if $p = y^3 + y$, drawn as left, then $py \cong y^4 + y^2$ can be drawn as right:



It just adds a default direction to each position. A copointed map from (q, ϵ) to (py, π) must pass the default direction back to the default direction in q, but leaves the other directions in p to go wherever they want to.

Example 7.22 (Slowing down dynamical systems). Given a dynamical system $f\colon Sy^S \to p$, we automatically get a map $Sy^S \to py$ to the cofree pointing. We called this "adding a pause button" in **??**. Thus we can take any dynamical system and replace it with one whose interface is copointed.

We can use a copointed interface to slow down a dynamical system, in a kind of inverse to how we sped up dynamical systems in (6.13). There we took a dynamical system f with interface p and produced one with interface $p^{\triangleleft k}$. Here we will take one with interface $p^{\triangleleft k}$ and produce one with interface p.

To do this, we need p to be copointed, i.e. we need to have in hand a map $\epsilon\colon p \to y$, and as we saw above that we can always assume that. Now for any $k \in \mathbb{N}$ we have k-many maps $p^{\triangleleft k} \to p$. For example, if $k = 3$, we have

$$\{\epsilon \triangleleft \epsilon \triangleleft p, \epsilon \triangleleft p \triangleleft \epsilon, p \triangleleft \epsilon \triangleleft \epsilon\} \subseteq \mathbf{Poly}(p^{\triangleleft 3}, p)$$

So given a dynamical system $Sy^S \to p^{\triangleleft k}$, which outputs as its position a whole k-fold strategy at one time and which takes as input sequences of k-many inputs, we can feed it one input and $k-1$ pauses. This is what you get when you compose $Sy^S \to p^{\triangleleft k} \to p$.

Given a dynamical system $f\colon Sy^S \to p$, where p is copointed and f preserves the copoint, we could speed it up as before to get $Sy^S \to p^{\triangleleft k}$ and then slow it down to get $Sy^S \to p$, and we get back f. So slowing down is a retract of speeding up in this sense.

Exercise 7.23 (Solution here).

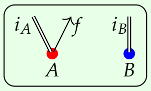
1. Show that there is a monoidal structure $(y, \otimes)$ on **Cpt**(**Poly**) such that the forgetful functor $U\colon \mathbf{Cpt}(\mathbf{Poly}) \to \mathbf{Poly}$ is strong monoidal.
2. Show that this monoidal structure is closed, i.e. that there is an internal hom $[-,-]$ on **Cpt**(**Poly**). Hint: you should have $U([py, qy]_{\mathbf{Cpt}}) \cong [py, q]_{\mathbf{Poly}}y$. ◊

7.3.2 Constructing the cofree comonoid

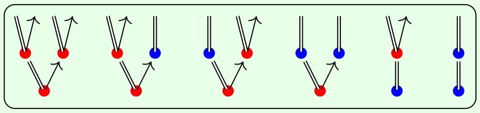
It remains to show that we can functorially take any copointed polynomial (q, ϵ) and return a comonoid, and that this construction is right adjoint to the forgetful functor. From the description in Section 7.2, we know the cofree comonoid is supposed to have something to do with infinite trees. And we know that the set of height-n p-trees is given by $p^{\triangleleft n}$. So we might think we can somehow take a limit of these height-n trees for various n.

The problem is there's no obvious maps between $p^{\triangleleft n}$ and $p^{\triangleleft n+1}$. Luckily, the copointing fixes that problem. Given $\epsilon\colon q \to y$, we have two maps $q \triangleleft q \rightrightarrows q$, namely $q \triangleleft \epsilon$ and $\epsilon \triangleleft q$.

Example 7.24. Suppose $q = \{A\}y^{\{i_A, f\}} + \{B\}y^{i_B}$ with copointing ϵ selecting i_A and i_B:

$q :=$ 

Then $q \triangleleft q$ looks as follows

$q \triangleleft q =$ 

How can we picture the maps $(q \triangleleft \epsilon), (\epsilon \triangleleft q)\colon q \triangleleft q \to q$?

The map $q \triangleleft \epsilon$ takes each position of $q \triangleleft q$ to whatever is on the bottom layer: it takes the first four to A and the last two to B. It passes back directions using the defaults (i_A and i_B) on the top layer.

The map $\epsilon \triangleleft q$ uses the defaults on the bottom layer instead. Every position in $q \triangleleft q$ has a default direction, and the corolla sitting there in the top layer is where $\epsilon \triangleleft q$ sends it, with identity on directions.

Indeed, for every n, there are $(n+1)$-many morphisms $q^{\triangleleft n+1} \to q^{\triangleleft n}$, so we have a diagram

$$\begin{tikzcd}
y & q \arrow[l, "\epsilon"'] & q \triangleleft q \arrow[l, shift right, "\epsilon \triangleleft q"'] \arrow[l, shift left, "q \triangleleft \epsilon"] & q^{\triangleleft 3} \arrow[l, shift right=2, "\epsilon \triangleleft q \triangleleft q"'] \arrow[l, "q \triangleleft \epsilon \triangleleft q" description] \arrow[l, shift left=2, "q \triangleleft q \triangleleft \epsilon"] & \cdots \arrow[l, shift right=3] \arrow[l, shift right=1] \arrow[l, shift left=1] \arrow[l, shift left=3]
\end{tikzcd} \tag{7.25}$$

The cofree comonoid is given by the limit of this diagram.

Let's denote the shape of this diagram by Δ_+: its objects are finite ordered sets—including the empty set—and its morphisms are order-preserving injections. For any copointed polynomial $q \xrightarrow{\epsilon} y$, we get a diagram $Q\colon \Delta_+ \to \mathbf{Poly}$ as above, and this is functorial in q.

Theorem 7.26. For any copointed polynomial $q \xrightarrow{\epsilon} y$, let $\bar{q}$ denote the limit of $Q\colon \Delta_+ \to$ **Poly**. It naturally has the structure of a comonoid $(\bar{q}, \epsilon, \delta)$, and this construction is right adjoint to the forgetful functor

$$\begin{tikzcd} \mathbf{Comon}(\mathbf{Poly}) \arrow[r, shift left=2, "U"] \arrow[r, phantom, "\Rightarrow"] & \mathbf{Cpt}(\mathbf{Poly}) \arrow[l, shift left=2, "\bar{q}"] \end{tikzcd} .$$

Proof sketch. We first give $\bar{q}$ the structure of a comonoid. Since $\bar{q}$ is the limit of (7.25), the inclusion the inclusion $\{0\} \to \Delta_+$ induces a projection map $\bar{q} \to y$, which we again call ϵ. Since $\triangleleft$ commutes with connected limits in both variables and Δ_+ is connected, we have that $\bar{q} \triangleleft \bar{q}$ is the limit of the following $\Delta_+ \times \Delta_+$-shaped diagram:

$$\bar{q} \triangleleft \bar{q} \cong \lim \left(\begin{tikzcd}
q^{\triangleleft(0+0)} & q^{\triangleleft(0+1)} \arrow[l] & q^{\triangleleft(0+2)} \arrow[l, shift left] \arrow[l, shift right] & q^{\triangleleft(0+3)} \arrow[l, shift left=2] \arrow[l] \arrow[l, shift right=2] & \cdots \arrow[l, shift left=3] \arrow[l, shift left=1] \arrow[l, shift right=1] \arrow[l, shift right=3] \\
q^{\triangleleft(1+0)} \arrow[u] & q^{\triangleleft(1+1)} \arrow[l] \arrow[u] & q^{\triangleleft(1+2)} \arrow[l, shift left] \arrow[l, shift right] \arrow[u] & q^{\triangleleft(1+3)} \arrow[l, shift left=2] \arrow[l] \arrow[l, shift right=2] \arrow[u] & \cdots \arrow[l, shift left=3] \arrow[l, shift left=1] \arrow[l, shift right=1] \arrow[l, shift right=3] \\
q^{\triangleleft(2+0)} \arrow[u, shift left] \arrow[u, shift right] & q^{\triangleleft(2+1)} \arrow[l] \arrow[u, shift left] \arrow[u, shift right] & q^{\triangleleft(2+2)} \arrow[l, shift left] \arrow[l, shift right] \arrow[u, shift left] \arrow[u, shift right] & q^{\triangleleft(2+3)} \arrow[l, shift left=2] \arrow[l] \arrow[l, shift right=2] \arrow[u, shift left] \arrow[u, shift right] & \cdots \arrow[l, shift left=3] \arrow[l, shift left=1] \arrow[l, shift right=1] \arrow[l, shift right=3] \\
q^{\triangleleft(3+0)} \arrow[u, shift left=2] \arrow[u] \arrow[u, shift right=2] & q^{\triangleleft(3+1)} \arrow[l] \arrow[u, shift left=2] \arrow[u] \arrow[u, shift right=2] & q^{\triangleleft(3+2)} \arrow[l, shift left] \arrow[l, shift right] \arrow[u, shift left=2] \arrow[u] \arrow[u, shift right=2] & q^{\triangleleft(3+3)} \arrow[l, shift left=2] \arrow[l] \arrow[l, shift right=2] \arrow[u, shift left=2] \arrow[u] \arrow[u, shift right=2] & \cdots \arrow[l, shift left=3] \arrow[l, shift left=1] \arrow[l, shift right=1] \arrow[l, shift right=3] \\
\vdots \arrow[u, shift left=3] \arrow[u, shift left=1] \arrow[u, shift right=1] \arrow[u, shift right=3] & \vdots \arrow[u, shift left=3] \arrow[u, shift left=1] \arrow[u, shift right=1] \arrow[u, shift right=3] & \vdots \arrow[u, shift left=3] \arrow[u, shift left=1] \arrow[u, shift right=1] \arrow[u, shift right=3] & \vdots \arrow[u, shift left=3] \arrow[u, shift left=1] \arrow[u, shift right=1] \arrow[u, shift right=3] & \ddots
\end{tikzcd} \right)$$

There is a commutative diagram in **Cat**

$$\begin{tikzcd}
\Delta_+ \times \Delta_+ \arrow[rr, "+"] \arrow[dr, bend right, "(m_1, m_2) \mapsto q^{\triangleleft(m_1+m_2)}"'] & & \Delta_+ \arrow[dl, bend left, "n \mapsto q^{\triangleleft n}"] \\
& \mathbf{Poly} &
\end{tikzcd} \tag{7.27}$$

which induces a map (in the opposite direction) between their limits $\delta\colon \bar{q} \to \bar{q} \triangleleft \bar{q}$, which we take to be the comultiplication. Appending (7.27) with the inclusion $\{0\} \times \Delta_+ \to \Delta_+ \times \Delta_+$, etc., it is easy to see that $(\bar{q}, \epsilon, \delta)$ satisfies the axioms of a comonoid.

We sketch the proof that this construction is right adjoint to the forgetful functor. For any copointed polynomial (q, ϵ), there is a counit map $\bar{q} \to q$, given by the obvious projection of the limit (7.25). Given a comonoid $(\mathfrak{c}, \epsilon, \delta)$, there is a morphism $\mathfrak{c} \to \bar{\mathfrak{c}}$

induced by the maps $\delta^{n-1}\colon \mathfrak{c} \to \mathfrak{c}^{\triangleleft n}$ from Example 6.4. It is easy to check that these commute with the ϵ's in the diagram. To see that $\mathfrak{c} \to \bar{\mathfrak{c}}$ extends to a morphism of comonoids amounts to checking that the diagram

$$\begin{tikzcd}
\mathfrak{c} \arrow[r, "\delta^{m+n-1}"] \arrow[d, "\delta"'] & \mathfrak{c}^{\triangleleft(m+n)} \arrow[d, equal] \\
\mathfrak{c} \triangleleft \mathfrak{c} \arrow[r, "\delta^{m-1} \triangleleft \delta^{n-1}"'] & \mathfrak{c}^{\triangleleft m} \triangleleft \mathfrak{c}^{\triangleleft n}
\end{tikzcd}$$

commutes for any $m, n \in \mathbb{N}$. Both triangle equations are straightforward. □

Remark 7.28. The construction of the cofree comonoid from a copointed endofunctor in the proof of Theorem 7.26 is fairly standard; see [Lac10]. Nelson Niu has also constructed the cofree comonoid on a polynomial using a different limit diagram

$$\begin{tikzcd}
y \arrow[d] & p \arrow[d] & p \triangleleft p \arrow[d] & p \triangleleft p \triangleleft p \arrow[d] & \cdots \\
1 & p \triangleleft 1 \arrow[l, "!"'] & p \triangleleft p \triangleleft 1 \arrow[l, "p \triangleleft !"'] & p \triangleleft p \triangleleft p \triangleleft 1 \arrow[l, "p \triangleleft p \triangleleft !"'] & \cdots \arrow[l]
\end{tikzcd}$$

in terms of the original polynomial p, rather than from its copointing py; that is, this construction is right adjoint to $\mathbf{Comon}(\mathbf{Poly}) \to \mathbf{Poly}$. One could also construct this right adjoint using the following limit, again applied to the original polynomial p:

$$\begin{tikzcd}
y \arrow[dr] & & p \arrow[dl] \arrow[dr] & & p \triangleleft p \arrow[dl] \arrow[dr] & & p \triangleleft p \triangleleft p \arrow[dl] \arrow[dr] & \cdots \\
& 1 & & p \triangleleft 1 & & p \triangleleft p \triangleleft 1 & & \cdots
\end{tikzcd}$$

We record the following proposition here; it will be useful in Corollary 10.7.

Proposition 7.29. If $f\colon p \to q$ is a Cartesian map of polynomials, then $\mathsf{tree}_f\colon \mathsf{tree}_p \to \mathsf{tree}_q$ is a Cartesian cofunctor. That is, for each tree $t \in \mathsf{tree}_p(1)$ the function $\mathsf{tree}_f^\sharp\colon \mathsf{tree}_p[t] \xrightarrow{\cong} \mathsf{tree}_q[\mathsf{tree}_f(t)]$ is a bijection.

Proof. ** □

Proposition 7.30. The cofree comonoid functor is lax monoidal; in particular, we have maps

$$y \to \mathscr{T}_y \qquad \text{and} \qquad \mathscr{T}_p \otimes \mathscr{T}_q \to \mathscr{T}_{p \otimes q}$$

for any $p, q \in \mathbf{Poly}$.

7.4 Sys(p) is a topos

Theorem 7.31. Let $\mathscr{T}_p$ be the cofree comonoid on $p \in \mathbf{Poly}$. There is an equivalence of categories

$$\mathbf{Sys}(p) \cong \mathbf{Cat}(\mathscr{T}_p, \mathbf{Set})$$

between the category of dynamical systems on p and that of functors $\mathscr{T}_p \to \mathbf{Set}$.

Proof. ** (In general, coalgebras of comonoids are copresheaves on the corresponding category) □

A consequence of this is that **Sys**(p) forms a topos, and hence has a ready-made type theory and internal logic. While we don't have space to do this justice, we will briefly discuss the sort of logical statement one can make about dynamical systems.

7.5 Morphisms between cofree comonoids

Given a morphism of polynomials $\varphi \colon p \to q$, the cofree functor gives us a map of comonoids $\mathscr{T}_\varphi \colon \mathscr{T}_p \to \mathscr{T}_q$, which works as follows.

An object $t \in \mathsf{tree}_p$ is a tree; the tree $u := \mathscr{T}_\varphi(t) \in \mathsf{tree}_q$ is constructed recursively as follows. If the root of t is $i \in p(1)$ then the root of u is $j := \varphi_1(i)$. To each branch $e \in q[j]$, we need to assign a new tree, and we use the one situated at $\varphi_i^\sharp(e)$.

Exercise 7.32 (Solution here). Let $p := y^2 + 1$ and $q := 2y + 2$.

1. Choose a map $\varphi \colon p \to q$, and write it out.
2. Choose a tree $t \in \mathsf{tree}_p$ with at least height 2.
3. What is $\mathscr{T}_\varphi(t)$? ◊

Exercise 7.33 (Solution here). The following exercise is useful when considering the (topos-theoretic) logic of dynamical systems. Namely, it will allow us to specify legal subtrees of height n.

1. Choose a polynomial q and a map $\epsilon \colon q \to y$, i.e. a copointed polynomial.
2. Recall from Theorem 7.26 that the carrier $\bar{q}$ of the cofree comonoid $\mathscr{T}_q = (\bar{q}, \epsilon, \delta)$ is constructed as a limit

$$\begin{tikzcd}
\bar{q} = \lim\Big(y & q \arrow[l] & q \triangleleft q \arrow[l, shift left] \arrow[l, shift right] & q^{\triangleleft 3} \arrow[l, shift left=2] \arrow[l] \arrow[l, shift right=2] & \cdots \Big) \arrow[l, shift left=3] \arrow[l, shift left=1] \arrow[l, shift right=1] \arrow[l, shift right=3]
\end{tikzcd}$$

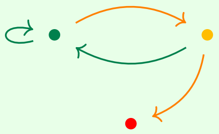
and in particular there is a structure map $\bar{q} \to q^{\triangleleft n}$, for any $n \in \mathbb{N}$. Where does it send a tree $t \in \mathsf{tree}_q$?
3. There is an induced cofunctor $\mathscr{T}_q \to \mathscr{T}_{q^{\triangleleft n}}$. Show that for all $n \geq 1$, it is it an isomorphism. ◊

7.6 Consequences of adjointness

Recall from Definition 3.17 that a dependent system (or generalized Moore machine) is a map of polynomials $f\colon Sy^S \to p$. Here S is called the set of states and p is the interface.

But now we know that Sy^S is secretly the underlying polynomial of a comonoid. This means that f has a mate, i.e. a corresponding cofunctor $Sy^S \nrightarrow \mathscr{T}_p$ to the category of p-trees. How does that work?

Example 7.34. Let $S :=$ {🟢, 🟡, 🔴} and $p := y^2 + y$, and consider the dynamical system $f\colon Sy^S \to p$ from Exercise 3.21, depicted here again for your convenience:

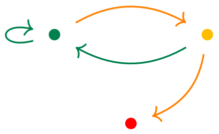


The polynomial map f is supposed to induce a cofunctor $F\colon Sy^S \nrightarrow \mathscr{T}_p$ from the state category on S to the category of p-trees. Thus to each state $s \in S$, we need to associate a tree; which should it be?

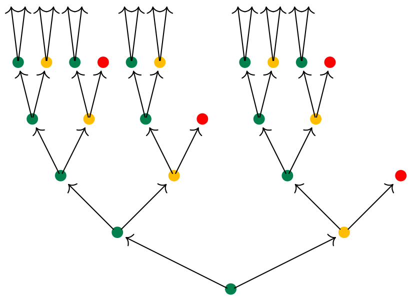
Exercise 7.35 (Solution here). Consider the walking arrow category $\mathcal{W} = \boxed{\bullet \to \bullet}$. Draw the cofunctor $\mathcal{W} \nrightarrow \mathscr{T}_{y^2+y}$. ◊

7.6.1 Dynamical systems and graph fibrations

In Example 7.34, there's a certain relationship we can see between the graph we associate to the dynamical system $Sy^S \to p$, namely



and the trees (which are also graphs) that its mate $Sy^S \to \mathscr{T}_p$ associates to each element of S, e.g. for the green dot:



Indeed, there is a map of graphs from the latter to the former, which sends all the green dots in the tree to the green dot in the dynamical system, etc. This map of graphs is a kind of *fibration*, or maybe we should say op-fibration, in the sense that the set of arrows emanating from every dot in the tree is in bijection with the set of arrows emanating from its image in the dynamical system graph.

Exercise 7.36 (Solution here).

1. Draw the other two trees associated to the dynamical system in Example 7.34.
2. Do they also have an op-fibration down to the dynamical system graph?
3. Are these op-fibrations special in any way? That is, are they unique, or have any universal property? ◊

7.6.2 Replacing Sy^S by another comonoid

For any interface p, we defined a dependent dynamical system—also called a generalized Moore machine—to be a set S and a polynomial map $Sy^S \to p$. But now it seems that what really makes this work is that Sy^S underlies a comonoid. This suggests that we could instead have defined a dependent dynamical system to be a comonoid $\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$ together with a map $\mathfrak{c} \to p$. What are the similarities and differences?

Here are some similarities. We still get a cofunctor $F\colon \mathcal{C} \to \mathscr{T}_p$, so we associate a p-tree to each object in $\mathcal{C}$ and pass back paths out of its root to morphisms in $\mathcal{C}$. In terms of dynamics, we would think of objects in $\mathcal{C}$ as internal states. We still have the situation from (**??**), meaning that for every state $c \in \mathcal{C}$, we get a position $i \coloneqq F_1(c)$ in $p(1)$, and for every direction $d \in p[i]$ there we get a new state $\operatorname{cod}(F_c^\sharp(d)) \in \mathcal{C}$.

But in fact we get a little more from F, and this is where the differences come in. Namely, given a direction $d \in p[i]$, we get the morphism $F_c^\sharp(d)$ itself. In the state category Sy^S, there is a unique morphism between every two objects, so this passed-back morphism carries no data beyond what its codomain is. But for a more general comonoid $\mathcal{C}$, the morphisms *do* carry data.

Thus we can think of a map $\mathfrak{c} \to p$ as a dynamical system that "records its history." That is, given a path $\mathscr{T}_p$, a sequence of inputs to our dynamical system, we get a morphism in $\mathcal{C}$. If, unlike in a state category Sy^S, there are multiple morphisms between objects, we will know which one was actually taken by the system.

This seems like a nice generalization of dynamical systems—history-recording dynamical systems—and may have some use. However, we will see in Part III that there are strong theoretical reasons to emphasize the ahistorical state categories Sy^S. For one thing, the category of all such Sy^S-style dynamical systems on p forms a topos for any $p \in \mathbf{Poly}$.

7.7 Some math about cofree comonoids

Proposition 7.37. For every polynomial p, the cofree category $\mathscr{T}_p$ is free on a graph. That is, there is a graph G_p whose associated free category in the usual sense (the category of vertices and paths in G_p) is isomorphic to $\mathscr{T}_p$.

Proof. For vertices, we let V_p denote the set of p-trees,

$$V_p \coloneqq \mathsf{tree}_p(1).$$

For arrows we use the map $\pi\colon \mathsf{tree}_p \to p$ from **??** to define

$$A_p \coloneqq \sum_{t \in \mathsf{tree}_p(1)} p[\pi_1(t)]$$

In other words A_p is the set $\{d \in p[\pi_1(t)] \mid t \in \mathsf{tree}_p\}$ of directions in p that emanate from the root corolla of each p-tree. The source of (t, d) is t and the target is $\mathrm{cod}(\pi_t^\sharp(d))$. It is clear that every morphism in $\mathscr{T}_t$ is the composite of a finite sequence of such morphisms, completing the proof. □

Corollary 7.38. Let p be a polynomial and $\mathscr{T}_p$ the cofree comonoid. Every morphism in $\mathcal{C}_p$ is both monic and epic.

Proof. The free category on a graph always has this property, so the result follows from Proposition 7.37. □

Proposition 7.39. The cofree functor $p \mapsto \mathscr{T}_p = (\mathsf{tree}_p, \mathrm{root}, \mathrm{focus})$ is lax monoidal; in particular there is a map of polynomials $y \to \mathsf{tree}_y$, and for any $p, q \in \mathbf{Poly}$ there is a natural map

$$\mathsf{tree}_p \otimes \mathsf{tree}_q \to \mathsf{tree}_{p \otimes q}.$$

satisfying the usual conditions.

Proof. By Proposition 6.104, the left adjoint $U\colon \mathbf{Cat}^\sharp \to \mathbf{Poly}$ is strong monoidal. A consequence of Kelly's doctrinal adjunction theorem [Kel74] says that the right adjoint of an op-lax monoidal functor is lax monoidal. □

Exercise 7.40 (Solution here).

1. What polynomial is tree_y?
2. What is the map $y \to \mathsf{tree}_y$ from Proposition 7.39?
3. Explain in words how to think about the map $\mathsf{tree}_p \otimes \mathsf{tree}_q \to \mathsf{tree}_{p \otimes q}$ from Proposition 7.39, for arbitrary $p, q \in \mathbf{Poly}$. ◊

Proposition 7.41. The additive monoid $y^{\mathbb{N}}$ of natural numbers has a ×-monoid structure in $\mathbf{Cat}^{\sharp}$.

Proof. The right adjoint $p \mapsto \mathscr{T}_p$ preserves products, so $y^{\mathsf{List(n)}} \cong \mathscr{T}_{y^{\mathsf{n}}}$ is the n-fold product of $y^{\mathbb{N}}$ in $\mathbf{Cat}^{\sharp}$. We thus want to find cofunctors $e\colon y \to y^{\mathbb{N}}$ and $m\colon y^{\mathsf{List(2)}} \to y^{\mathbb{N}}$ that satisfy the axioms of a monoid.

The unique polynomial map $y \to y^{\mathbb{N}}$ is a cofunctor (it is the mate of the identity $y \to y$). We take m to be the mate of the polynomial map $y^{\mathsf{List(2)}} \to y$ given by the list $[1, 2]$. One can check by hand that these definitions make $(y^{\mathbb{N}}, e, m)$ a monoid in $(\mathbf{Cat}^{\sharp}, y, \times)$. □

Recall from Example 6.20 that an admissible section of a category $\mathcal{C}$ is a cofunctor $\mathcal{C} \nrightarrow y^{\mathbb{N}}$.

Corollary 7.42. For any category $\mathcal{C}$, the set $\mathbf{Cat}^{\sharp}(\mathcal{C}, y^{\mathbb{N}})$ of admissible sections has the structure of a monoid. Moreover, this construction is functorial

$$\mathbf{Cat}^{\sharp}(-, y^{\mathbb{N}})\colon \mathbf{Cat}^{\sharp} \to \mathbf{Mon}^{\mathrm{op}}$$

Proof. We saw that $y^{\mathbb{N}}$ is a monoid object in Proposition 7.41. □

A cofunctor $\mathcal{C} \nrightarrow y^{\mathbb{N}}$ is a policy in $\mathcal{C}$: it assigns an outgoing morphism to each object of $\mathcal{C}$. Any two such trajectories can be multiplied: we simply do one and then the other; this is the monoid operation. The policy assigning the identity to each object is the unit of the monoid.

We use the notation $\mathcal{C} \mapsto \vec{\mathcal{C}}$ for the monoid of admissible sections.

Theorem 7.43. The admissible sections functor

$$\mathbf{Cat}^{\sharp} \to \mathbf{Mon}^{\mathrm{op}}$$

is right adjoint to the inclusion $\mathbf{Mon}^{\mathrm{op}} \to \mathbf{Cat}^{\sharp}$ from Proposition 6.50.

Proof. Let $\mathcal{C}$ be a category and $(M, e, *)$ a monoid. A cofunctor $F\colon \mathcal{C} \nrightarrow y^M$ has no data on objects; it is just a way to assign to each $c \in \mathcal{C}$ and $m \in M$ a morphism $F_c^{\sharp}(m)\colon c \to c'$ for some $c' := \operatorname{cod}(F_c^{\sharp}(m))$. This assignment must send identities to identities and composites to composites: given $m' \in M$ we have $F_c^{\sharp}(m \mathbin{;} m') = F_c^{\sharp}(m) \mathbin{;} F_{c'}^{\sharp}(m')$. This is exactly the data of a monoid morphism $M \to \vec{\mathcal{C}}$: it assigns to each $m \in M$ an admissible section $\mathcal{C}$, preserving unit and multiplication. □

Proposition 7.44. There is a commutative square of left adjoints

$$\begin{tikzcd}
\mathbf{Mon}^{\mathrm{op}} \arrow[r, "U"] \arrow[d, "y^{-}"'] & \mathbf{Set}^{\mathrm{op}} \arrow[d, "y^{-}"] \\
\mathbf{Cat}^{\sharp} \arrow[r, "U"'] & \mathbf{Poly}
\end{tikzcd}$$

where the functors denoted U are forgetful functors.

Proof. Using the fully faithful functor $y^{-}\colon \mathbf{Mon}^{\mathrm{op}} \leftrightarrows \mathbf{Cat}^{\sharp}$ from Proposition 6.50, it is easy to check that the above diagram commutes.

The free-forgetful adjunction $\mathbf{Set} \leftrightarrows \mathbf{Mon}$ gives an opposite adjunction $\mathbf{Set}^{\mathrm{op}} \leftrightarrows \mathbf{Mon}^{\mathrm{op}}$, where U is now left adjoint. We saw that $y^{-}\colon \mathbf{Set}^{\mathrm{op}} \to \mathbf{Poly}$ is a left adjoint in Proposition 4.12, that $U\colon \mathbf{Cat}^{\sharp} \to \mathbf{Poly}$ is a left adjoint in Theorem 7.1, and that $y^{-}\colon \mathbf{Mon} \to \mathbf{Cat}^{\sharp}$ is a left adjoint in Theorem 7.43. □

7.8 Exercise solutions

Solution to Exercise 7.16.

Take $p := 2y$, and consider the object $x \in \mathscr{T}_p(1)$ given by the stream

$$x := (2\,12\,112\,1112\,11112\,111112\ldots)$$

(with spaces only for readability); note that every morphism emanating from x has a different codomain. We need to give $\varphi_i^{\sharp}(q)$ for every $i \in \mathscr{T}_p(1)$ and $q \geq 0$. Define

$$\varphi_i^{\sharp}(q) := \begin{cases} i & \text{if } i \neq x \text{ or } q = 0 \\ x' & \text{if } i = x \text{ and } q > 0 \end{cases}$$

where $x' := (12\,112\,1112\,11112\,111112\ldots)$. There are three cofunctor conditions to check, namely identity, codomains, and composition. The codomain condition is vacuous since $y^{\mathbb{Q}}$ has one object, and the identity condition holds by construction, because we always have $\varphi_i^{\sharp}(0) = i$. Now take $q_1, q_2 \in Q$; we need to check that

$$\varphi^{\sharp}_{\operatorname{cod}\varphi_i^{\sharp}(q_1)}(q_2) \overset{?}{=} \varphi_i^{\sharp}(q_1 + q_2)$$

holds. If $i \neq x$ or $q_1 = q_2 = 0$, then it holds because both sides equal i. If $i = x$ and either $q_1 > 0$ or $q_2 > 0$, it is easy to check that both sides equal x', so again it holds.

Part III

Data dynamics

Chapter 8

Copresheaves, databases, and dynamical systems

In the previous part we saw that comonoids in **Poly** are categories, but that morphisms of comonoids are not functors; they're called cofunctors. If someone were to ask "which are better, functors or cofunctors," the answer is clear. Functors are fundamental to mathematics itself, relating branches from set theory to logic to algebra to measure theory, etc. Cofunctors don't have anywhere near that sort of reach in terms of applicability. Still they provide an interesting way to compare categories, as well as new invariants of categories, like the monoid of direction fields on a category.

In this part we'll consider another kind of morphism between comonoids in **Poly**, i.e. categories, and one that is a bit more familiar than cofunctors. Namely, we'll consider the bimodules between comonoids. One might want to call them bi-co-modules, but this name is just a bit too long and the name bimodule is not ambiguous, so we'll go with it. So why do I say they're more familiar?

It turns out that bimodules between comonoids in **Poly** (categories) are also important objects of study in category theory. If $\mathcal{C}$ and $\mathcal{D}$ are categories, Richard Garner showed that a bimodule between them can be identified with what's known as a *parametric right adjoint* between the associated copresheaf categories $\mathcal{C}$-**Set** and $\mathcal{D}$-**Set**. Parametric right adjoints, or *pra*'s come up in ∞-category theory, but they also have a much more practical usage: they are the so-called *data migration functors*.

Indeed, we'll make due on a claim we made in **??**, that there is a strong connection between the basic theory of databases and the theory of bimodules in **Poly**. Databases have two parts: they have a schema, a specification of various types and relationships between them, and an instance, which is actual data sorted into those types and having those relationships. As we'll see, one can formalize the schema as a category $\mathcal{C}$ and the instance as a functor $I\colon \mathcal{C} \to \mathbf{Set}$. Here's an example of a theorem we'll prove:

Theorem 8.1. For a comonoid $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$, also understood as a category $\mathcal{C}$, the following categories are equivalent:

1. functors $\mathcal{C} \to \mathbf{Set}$;
2. discrete opfibrations over $\mathcal{C}$;
3. cartesian cofunctors to $\mathscr{C}$;
4. linear left $\mathscr{C}$-modules;
5. constant left $\mathscr{C}$-modules;
6. $(\mathscr{C}, 0)$-bimodules;
7. representable right $\mathscr{C}$-modules;
8. $\mathscr{C}$-coalgebras (sets with a coaction by $\mathscr{C}$).

Moreover, up to isomorphism, a $\mathscr{C}$-coalgebra can be identified with a dynamical system with comonoid interface $\mathscr{C}$.

The proof will be given in Theorem 9.9.

The plan of the part is as follows. We'll begin in Chapter 8 by reviewing copresheaves on a category, and their relationship to databases and dynamical systems. Then in Chapter 9 we'll prove a number of theoretical results, including Theorem 9.9 and Garner's "bimodules are parametric right adjoints" result. We'll continue to give intuition and applications in database and dynamical systems theory. Finally in Section 10.3 we'll provide some looser discussion and lay out some open questions.

Let $\mathcal{C}$ be a small category. One of the most important constructions in category theory is that of the category of copresheaves on $\mathcal{C}$.[1] This is the category

$$\mathcal{C}\text{-}\mathbf{Set} \coloneqq \mathbf{Fun}(\mathcal{C}, \mathbf{Set})$$

whose objects are functors $\mathcal{C} \to \mathbf{Set}$ and whose morphisms are natural transformations between them.

Example 8.2. Suppose $(G, e, *)$ is a monoid (e.g. a group). In a first course on abstract algebra, one encounters the notion of a *G-set*, which is a set X together with a G action: for every element $g \in G$ we get a function $\alpha_g \colon X \to X$; we might write $\alpha_g(x)$ as $g \cdot x$. To be a G-action, the $\cdot$ operation needs to satisfy two rules: $e \cdot x = x$ and $g \cdot (h \cdot x) = (g * h) \cdot x$. A morphism between two G-sets (sets X and Y, each equipped with a G-action) is just a function $f \colon X \to Y$ that satisfies a single rule: $f(g \cdot x) = g \cdot (f(x))$ for all $g \in G$ and $x \in X$.

Now recall that any monoid (e.g. a group) G can be understood as a category with one object, let's call our category $\mathcal{G}$ and the unique object $\blacktriangle$. The elements of G, including the identity and the multiplication, are encoded as the morphisms $\blacktriangle \to \blacktriangle$ in $\mathcal{G}$.

It turns out that G-sets are precisely functors $F \colon \mathcal{G} \to \mathbf{Set}$: the set $F(\blacktriangle)$ is our X

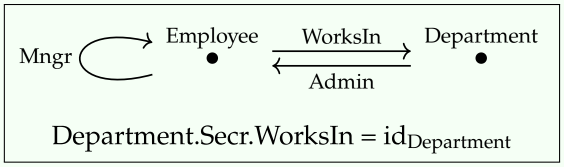
[1] Many would say that presheaves on $\mathcal{C}$ are more fundamental, but since the notions are equivalent—just use $\mathcal{C}^{\text{op}}$ to switch between them—we will consider the difference moot. We will focus on copresheaves.

above, and since the elements of G are now morphisms $g\colon \blacktriangle \to \blacktriangle$, the functor F sends them to functions $F(g)\colon X \to X$; this is the $g \cdot -$ operation. The axioms of a functor—preservation of identities and compositions—ensure the two rules of the $\cdot$ operation. Finally, morphisms between G-sets are exactly the natural transformations between functors; the naturality condition becomes the rule $f(g \cdot x) = g \cdot (f(x))$ we saw above.

G-sets are copresheaves on the associated one-object category $\mathcal{G}$.

Exercise 8.3 (Solution here). Consider the category $\mathcal{C}$ shown here:

$\mathcal{C} :=$



It is generated by two objects, three morphisms, and the equation shown above.

1. What is its emanation polynomial?

Consider now the database instance shown here:

Employee	**WorksIn**	**Mngr**
Alice	IT	Alice
Bobby	IT	Alice
Carla	Sales	Carla

Department	**Admin**
IT	Bobby
Sales	Carla

Consider this database instance as a functor $S\colon \mathcal{C} \to \mathbf{Set}$.

2. Say what sets S assigns the two objects.
3. Say what functions S assigns the three generating morphisms. ◊

Definition 8.4 (Discrete opfibration)**.** Let $\mathcal{C}$ be a category. A pair $(\mathcal{S}, \pi)$, where $\mathcal{S}$ is a category and $\pi\colon \mathcal{S} \to \mathcal{C}$ is a functor, is called a *discrete opfibration over* $\mathcal{C}$ if it satisfies the following condition.

- for every object $s \in \mathcal{S}$, object $c' \in \mathcal{C}$, and morphism $f\colon \pi(s) \to c'$ there exists a unique object $s' \in \mathcal{S}$ and morphism $\bar{f}\colon s \to s'$ such that $\pi(s') = c'$ and $\pi(\bar{f}) = f$.

$$\begin{tikzcd}
s \arrow[r, dashed, "\bar{f}"] \arrow[d, mapsto, "\pi"'] & s' \arrow[d, mapsto, "\pi"] \\
\pi(s) \arrow[r, "f"'] & c'
\end{tikzcd}$$

A *morphism* $(\mathcal{S}, \pi) \to (\mathcal{S}', \pi')$ between discrete opfibrations over $\mathcal{C}$ is a functor $F\colon \mathcal{S} \to$

$\mathcal{S}'$ making the following triangle commute:

$$\begin{tikzcd}
\mathcal{S} \arrow[rr, "F"] \arrow[dr, "\pi"'] & & \mathcal{S}' \arrow[dl, "\pi'"] \\
& \mathcal{C} &
\end{tikzcd} \tag{8.5}$$

We denote the category of discrete opfibrations into $\mathcal{C}$ by $\mathbf{dopf}(\mathcal{C})$.

Exercise 8.6 (Solution here). Show that if $F\colon \mathcal{S} \to \mathcal{S}'$ is a functor making the triangle (8.5) commute, then F is also a discrete opfibration. ◊

Exercise 8.7 (Solution here). Suppose $\pi\colon \mathcal{S} \to \mathcal{C}$ is a discrete opfibration and $i \in \mathcal{S}$ is an object. With notation as in Definition 8.4, show the following:

1. Show that the lift $\overline{\mathrm{id}_{\pi(i)}} = \mathrm{id}_i$ of the identity on $\pi(i)$ is the identity on i.
2. Show that for $f\colon \pi(i) \to c$ and $g\colon c \to c'$, we have $\bar{f} \fatsemi \bar{g} = \overline{f \fatsemi g}$.
3. Show that π is a cofunctor. ◊

Definition 8.8 (Category of elements $\int^{\mathcal{C}} I$)**.** Given a functor $I\colon \mathcal{C} \to \mathbf{Set}$, its category of elements $\int^{\mathcal{C}} I$ is defined to have objects

$$\mathrm{Ob}(\textstyle\int^{\mathcal{C}} I) \coloneqq \{(c, x) \mid c \in \mathrm{Ob}(\mathcal{C}), x \in I(c)\}$$

and given two objects (c, x) and (c', x'), the hom-set is given by

$$\mathrm{Hom}((c, x), (c', x')) \coloneqq \{f\colon c \to c' \mid I(f)(x) = x'\}.$$

Identities and composites in $(\int^{\mathcal{C}} I)$ are inherited from $\mathcal{C}$.

There is a functor $\pi\colon (\int^{\mathcal{C}} I) \to \mathcal{C}$ sending $\pi(c, x) \coloneqq c$ and $\pi(f) \coloneqq f$.

Exercise 8.9 (Solution here). Show that if $I\colon \mathcal{C} \to \mathbf{Set}$ is a functor then the functor $\pi\colon (\int^{\mathcal{C}} I) \to \mathcal{C}$ defined in Definition 8.8 is a discrete opfibration, as in Definition 8.4. ◊

Exercise 8.10 (Solution here). Draw the category of elements for the functor $S\colon \mathcal{C} \to \mathbf{Set}$ shown in Exercise 8.3. ◊

Exercise 8.11 (Solution here). Suppose that $I, J\colon \mathcal{C} \to \mathbf{Set}$ are functors and $\alpha\colon I \to J$ is a natural transformation.

1. Show that α induces a functor $(\int^{\mathcal{C}} I) \to (\int^{\mathcal{C}} J)$.
2. Show that it is a morphism of discrete opfibrations in the sense of Definition 8.4.
3. Show that this construction is functorial. We denote this functor by
$$\int^{\mathcal{C}}\colon \mathcal{C}\text{-}\mathbf{Set} \to \mathbf{dopf}(\mathcal{C})$$ ◊

Exercise 8.12 (Solution here). Let G be a graph, and let $\mathcal{G}$ be the free category on it. Show that for any functor $S\colon \mathcal{G} \to \mathbf{Set}$, the category $\int^{\mathcal{G}} S$ of elements is again free on a graph. ◊

Proposition 8.13. Let $\mathcal{C}$ be a category. The following are equivalent:

1. the category $\mathcal{C}$-**Set** of functors $\mathcal{C} \to \mathbf{Set}$,
2. the category of discrete opfibrations over $\mathcal{C}$,
3. the category of cartesian cofunctors into $\mathcal{C}$.

In fact the latter two are isomorphic.

Proof. By Exercise 8.11 we have a functor $\int^{\mathcal{C}}\colon \mathcal{C}\text{-}\mathbf{Set} \to \mathbf{dopf}(\mathcal{C})$. There is a functor going back: given a discrete opfibration $\pi\colon \mathcal{S} \to \mathcal{C}$, we define $(\partial\pi)\colon \mathcal{C} \to \mathbf{Set}$ on $c \in \mathrm{Ob}(\mathcal{C})$ and $f\colon c \to c'$ by

$$\begin{aligned}(\partial\pi)(c) &\coloneqq \{s \in \mathcal{S} \mid \pi(s) = c\}\\ (\partial\pi)(f)(s) &\coloneqq \bar{f}(s).\end{aligned}$$

On objects, the roundtrip $\mathcal{C}$-**Set** → $\mathcal{C}$-**Set** sends $I\colon \mathcal{C} \to \mathbf{Set}$ to the functor

$$\begin{aligned}c \mapsto \{s \in \int^{\mathcal{C}} I \mid \pi(s)\\ = \{(c, x) \mid x \in I(c)\} && = I(c).\end{aligned}$$

The roundtrip $\mathbf{dopf}(\mathcal{C}) \to \mathbf{dopf}(\mathcal{C})$ sends $\pi\colon \mathcal{S} \to \mathcal{C}$ to the discrete opfibration whose object set is $\{(c, s) \in \mathrm{Ob}(\mathcal{C}) \times \mathrm{Ob}(\mathcal{S}) \mid \pi(s) = c\}$ and this set is clearly in bijection with $\mathrm{Ob}(\mathcal{S})$. Proceeding similarly, one defines an isomorphism of categories $\mathcal{S} \cong \int^{\mathcal{C}} \partial\pi$.

The above correspondence is well-known; it remains to address the relationship between (2) and (3). A cartesian cofunctor $(\varphi_1, \varphi^\sharp)\colon \mathcal{S} \nrightarrow \mathcal{C}$ gives a function $\varphi\colon \mathrm{Ob}(\mathcal{S}) \to \mathrm{Ob}(\mathcal{C})$ and for each $s \in \mathcal{S}$ an isomorphism

$$\varphi^\sharp_s\colon \mathcal{C}[\varphi_1(s)] \xrightarrow{\cong} \mathcal{S}[s]$$

between the set of $\mathcal{S}$-morphisms emanating from s and the set of $\mathcal{C}$-morphisms emanating from $\varphi_1(s)$. This isomorphism respects identities, codomains, and composites. As such we can define a functor that acts as φ_1 on objects and $(\varphi^\sharp)^{-1}$ on morphisms, and it is easily checked to be a discrete opfibration.

Finally, given a discrete opfibration $\pi\colon \mathcal{S} \to \mathcal{C}$, we define $\varphi_1 := \mathrm{Ob}(\pi)$ to be its on-objects part, and for any $s \in \mathrm{Ob}(\mathcal{S})$ and emanating morphism $f \in \mathcal{C}[\varphi_1(s)]$, we define $\varphi_s^\sharp(f) := \bar{f}$ to be the lift guaranteed by Definition 8.4. The conversions between discrete opfibrations and cartesian cofunctors are easily seen to be functorial and the roundtrips are identities. □

Recall from **??** that a dynamical system on a category $\mathcal{C}$ consists of a set S and a cofunctor $Sy^S \nrightarrow \mathcal{C}$ from the state category on S to $\mathcal{C}$.

Proposition 8.14 (Database instances are dynamical systems)**.** Up to isomorphism, discrete opfibrations into $\mathcal{C}$ can be identified with dynamical systems on $\mathcal{C}$.

In case it isn't clear, this association is only functorial on the groupoid of objects and isomorphisms.

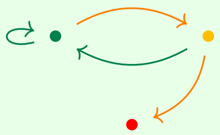
Proof. Given a discrete opfibration $\pi\colon \mathcal{S} \to \mathcal{C}$, take $S := \mathrm{Ob}(\mathcal{S})$ and define $(\varphi_1, \varphi^\sharp)\colon Sy^S \to \mathfrak{c}$ by $\varphi_1 = \pi$ and with $\varphi^\sharp$ given by the lifting: $\varphi(g) := \hat{g}$ as in Definition 8.4. One checks using Exercise 8.7 that this defines a cofunctor.

Conversely, given a cofunctor $(\varphi_1, \varphi^\sharp)\colon Sy^S \to \mathfrak{c}$, the function φ_1 induces a map of polynomials $Sy \to \mathfrak{c}$, and we can factor it as a vertical followed by a cartesian $Sy \to \mathfrak{s} \xrightarrow{\psi} \mathfrak{c}$. We can give $\mathfrak{s}$ the structure of a category such that ψ is a cofunctor; see Exercise 8.15. □

Exercise 8.15 (Solution here). With notation as in Proposition 8.14, complete the proof as follows.

1. Check that $(\varphi, \varphi^\sharp)$ defined in the first paragraph is indeed a cofunctor.
2. Find a comonoid structure on $\mathfrak{s}$ such that ψ is a cofunctor, as stated in the second paragraph.
3. Show that the two directions are inverse, up to isomorphism. ◊

Example 8.16. In Example 7.34 we had a dynamical system with $S := \{\bullet, \bullet, \bullet\}$ and $p := y^2 + y$, and $f\colon Sy^S \to p$ from Exercise 3.21, depicted here again for your convenience:

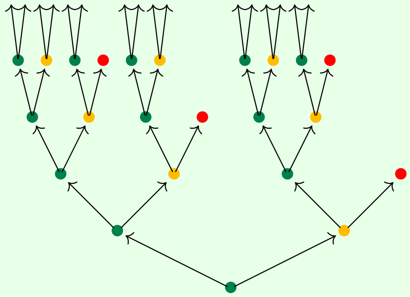


(8.17)

The polynomial map f induces a cofunctor $F\colon Sy^S \nrightarrow \mathscr{T}_p$ from the state category on S to the category of p-trees. We can now see this as a database instance on $\mathscr{T}_p$, considered as a database schema.

The cofree category $\mathscr{T}_t$ is actually the free category on a graph, as we saw in

Proposition 7.37, and so the schema is easy. There is one table for each tree (object in $\mathscr{T}_p$), e.g. we have a table associated to this tree:



The table has two columns, say left and right, corresponding to the two arrows emanating from the root node. The left column refers back to the same table, and the right column refers to another table (the one corresponding to the yellow dot).

Again, there are infinitely many tables in this schema. Only three of them have data in them; the rest are empty. We know in advance that this instance has three rows in total, since $|S| = 3$.

Given a dynamical system $Sy^S \to p$, we extend it to a cofunctor $\varphi\colon Sy^S \nrightarrow \mathscr{T}_p$. By Propositions 8.13 and 8.14, we can consider it as a discrete opfibration over $\mathscr{T}_p$. By Exercise 8.12 the category $\int \varphi$ is again free on a graph. It is this graph that we usually draw when depicting the dynamical system, e.g. in (8.17).

Exercise 8.18 (Solution here). Give an example of a dynamical system on $p \coloneqq y^2 + y$ for which the corresponding database instance has a table with at least two rows. ◊

Exercise 8.19 (Solution here). A dynamical system with interface y is a map $Sy^S \to y$.

1. What is the corresponding database schema?
2. Explain what the corresponding database instance looks like.
3. In particular, how many total rows does it have?
4. Give an example with $|S| = 7$, displayed both as a dynamical system (with states and transitions) and as a database instance. ◊

8.1 Exercise solutions

Chapter 9

Bimodules over polynomial comonoids

One might think that the database story—or you could call it the copresheaves story—is somewhat tacked on to the main line here: cartesian cofunctors $\mathcal{S} \nrightarrow \mathcal{C}$ can be seen as database instances on $\mathcal{C}$, but perhaps this is just an incidental curiosity. In this chapter we'll see that in fact copresheaves, i.e. set-valued functors on categories, are a major aspect of the story.

In a monoidal category, one can not only consider monoids and comonoids, but also modules between these. For example, in the monoidal category of abelian groups under bilinear product, the monoid objects are rings and the bimodules are bimodules in the usual sense. Here we are interested in comonoids rather than monoids, so we should be interested in bi-comodules; we will just call these bimodules for linguistic convenience.

We will see that bimodules in **Poly** are equivalent to data-migration functors. Given comonoids/categories $\mathcal{C}$ and $\mathcal{D}$, a bimodule $\mathcal{C} \overset{m}{\triangleleft\!\!\longleftarrow\!\!\triangleleft} \mathcal{D}$ is a way of moving database instances on $\mathcal{D}$ to database instances on $\mathcal{C}$. One could call it a "$\mathcal{D}$-indexed union of conjunctive queries."

Definition 9.1 (Bimodule)**.** Let $\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$ and $\mathcal{D} = (\mathfrak{d}, \epsilon, \delta)$ be comonoids. A $(\mathcal{C}, \mathcal{D})$-*bimodule* (m, λ, ρ), denoted $\mathcal{C} \overset{m}{\triangleleft\!\!\longleftarrow\!\!\triangleleft} \mathcal{D}$ or in shorthand $\mathfrak{c} \overset{m}{\triangleleft\!\!\longleftarrow\!\!\triangleleft} \mathfrak{d}$,[a] consists of

1. a polynomial $m \in \mathbf{Poly}$,
2. a morphism $\mathfrak{c} \triangleleft m \xleftarrow{\lambda} m$, and
3. a morphism $m \xrightarrow{\rho} m \triangleleft \mathfrak{d}$,

satisfying the following commutative diagrams:

$$\begin{tikzcd}
\mathfrak{c} \triangleleft m \arrow[d, "\epsilon \triangleleft m"'] & m \arrow[l, "\lambda"'] \arrow[dl, equal, bend left] \\
m &
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathfrak{c} \triangleleft m \arrow[d, "\delta \triangleleft m"'] & m \arrow[l, "\lambda"'] \arrow[d, "\lambda"] \\
\mathfrak{c} \triangleleft \mathfrak{c} \triangleleft m & \mathfrak{c} \triangleleft m \arrow[l, "\mathfrak{c} \triangleleft \lambda"]
\end{tikzcd}
\tag{9.2}$$

$$\begin{tikzcd}
m \arrow[r, "\rho"] \arrow[dr, equal, bend right] & m \triangleleft \mathfrak{d} \arrow[d, "m \triangleleft \epsilon"] \\
& m
\end{tikzcd}
\qquad
\begin{tikzcd}
m \arrow[r, "\rho"] \arrow[d, "\rho"'] & m \triangleleft \mathfrak{d} \arrow[d, "m \triangleleft \delta"] \\
m \triangleleft \mathfrak{d} \arrow[r, "\rho \triangleleft \mathfrak{d}"'] & m \triangleleft \mathfrak{d} \triangleleft \mathfrak{d}
\end{tikzcd}
\tag{9.3}$$

$$\begin{tikzcd}
m \arrow[r, "\rho"] \arrow[d, "\lambda"'] & m \triangleleft \mathfrak{d} \arrow[d, "\lambda \triangleleft \mathfrak{d}"] \\
\mathfrak{c} \triangleleft m \arrow[r, "\mathfrak{c} \triangleleft \rho"'] & \mathfrak{c} \triangleleft m \triangleleft \mathfrak{d}
\end{tikzcd}
\tag{9.4}$$

Just (m, λ) and the first line of diagrams is called a left $\mathcal{C}$-module, and similarly just (m, ρ) and the second line of diagrams is called a right $\mathfrak{D}$-module.

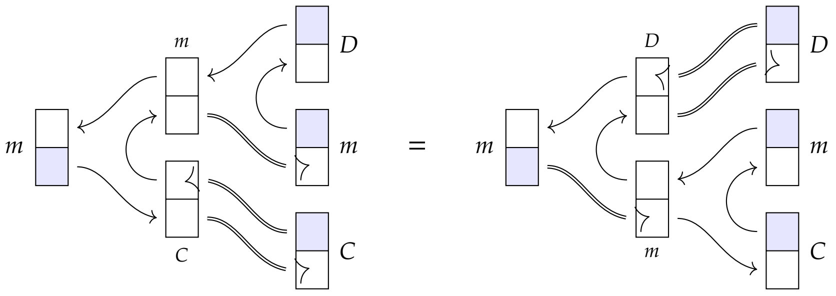
A *morphism* of $(\mathcal{C}, \mathcal{D})$-bimodules is a map $m \to n$ making the following diagram commute:

$$\begin{tikzcd}
\mathfrak{c} \triangleleft m \arrow[d] & m \arrow[l] \arrow[r] \arrow[d] & m \triangleleft \mathfrak{d} \arrow[d] \\
\mathfrak{c} \triangleleft n & n \arrow[l] \arrow[r] & n \triangleleft \mathfrak{d}
\end{tikzcd}$$

We denote the category of $(\mathcal{C}, \mathcal{D})$-bimodules by ${}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}}$.

[a] Note that the symbol $\mathfrak{c} \mathbin{\triangleleft\!\!\relbar\!\!\relbar\!\!\triangleleft} \mathfrak{d}$ looks like it's going backwards, from $\mathfrak{d}$ to $\mathfrak{c}$; this is good because we'll see that this is the direction of data migration for a $(\mathcal{C}, \mathcal{D})$-bimodule. But the symbology is also mnemonically good because the $\triangleleft$'s go in the correct direction $\mathfrak{c} \triangleleft m \leftarrow m$ and $m \to m \triangleleft \mathfrak{d}$.

We draw the commutativity of Eq. (9.4) using polyboxes.

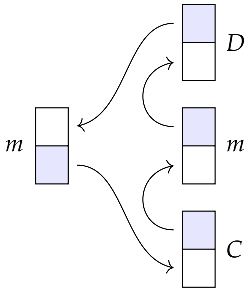


(9.5)

Exercise 9.6 (Solution here).

1. Draw the equations of Eq. (9.2) using polyboxes.
2. Draw the equations of Eq. (9.3) using polyboxes. ◊

Note that Eq. (9.5) means that we can unambiguously write



Exercise 9.7 (Solution here). Let $\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$ be a category. Recall from **??** that y has a unique category structure.

1. Show that a left $\mathcal{C}$-module is the same thing as a $(\mathcal{C}, y)$-bimodule.
2. Show that a right $\mathcal{C}$-module is the same thing as a $(y, \mathcal{C})$-bimodule.
3. Show that every polynomial $p \in \mathbf{Poly}$ has a unique (y, y)-bimodule structure.
4. Show that there is an isomorphism of categories $\mathbf{Poly} \cong {}_y\mathbf{Mod}_y$. ◊

Definition 9.8 ($\mathscr{C}$-coalgebra)**.** Let $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ be a comonoid. A *$\mathscr{C}$-coalgebra* consists of a set S and a function $\alpha\colon S \to \mathfrak{c} \triangleleft S$, making the two diagrams below commute:

$$\begin{tikzcd}
\mathfrak{c} \triangleleft S \arrow[d, "\epsilon\triangleleft"'] & S \arrow[l, "\alpha"'] \arrow[dl, equal, bend left] \\
S &
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathfrak{c} \triangleleft S \arrow[d, "\delta\triangleleft S"'] & S \arrow[l, "\alpha"'] \arrow[d, "\alpha"] \\
\mathfrak{c} \triangleleft \mathfrak{c} \triangleleft m & \mathfrak{c} \triangleleft S \arrow[l, "\mathfrak{c}\triangleleft\alpha"]
\end{tikzcd}$$

In other words, it is a left $\mathfrak{c}$-comodule whose carrier is constant on a set. A morphism is a function $S \to T$ commuting with α, just as for comodules; see Definition 9.1.

In order to as quickly as possible orient the reader to bimodules, we begin by proving the following.

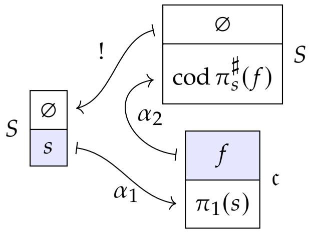
Theorem 9.9. For a comonoid $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ (category $\mathcal{C}$), the following categories are equivalent:

1. functors $\mathcal{C} \to \mathbf{Set}$;
2. discrete opfibrations over $\mathcal{C}$;
3. cartesian cofunctors to $\mathscr{C}$;
4. $\mathscr{C}$-coalgebras (sets with a coaction by $\mathscr{C}$);
5. constant left $\mathscr{C}$-modules;
6. $(\mathscr{C}, 0)$-bimodules;
7. linear left $\mathscr{C}$-modules; and
8. representable right $\mathscr{C}$-modules (opposite).

In fact, all but the first are isomorphic categories, and their core groupoid of is that of dynamical systems with interface $\mathcal{C}$.

Proof. $1 \simeq 2 \cong 3$**:** This was shown in Proposition 8.13.

$3 \cong 4$**:** Given a cartesian cofunctor $(\pi_1, \pi^\sharp)\colon \mathcal{S} \nrightarrow \mathcal{C}$, let $S := \mathrm{Ob}(\mathcal{S})$ and define a $\mathfrak{c}$-coalgebra structure $\alpha\colon S \to \mathfrak{c} \triangleleft S$ on an object $s \in \mathrm{Ob}(\mathcal{S})$ and an emanating morphism $f\colon \pi_1(s) \to c'$ in $\mathcal{C}$ by



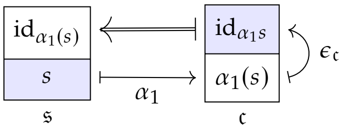
We check that this is indeed a coalgebra using properties of cofunctors. For identities in $\mathscr{C}$, we have

$$\begin{aligned}\alpha_2(s, \mathrm{id}_{\pi_1(s)}) &= \operatorname{cod}\pi^\sharp_s(\mathrm{id}_{\pi_1(s)})\\ &= \operatorname{cod}\mathrm{id}_s = s\end{aligned}$$

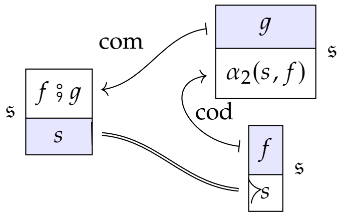
and for compositions in $\mathscr{C}$ we have

$$\begin{aligned}\alpha_2(s, f \fatsemi g) &= \operatorname{cod}\left(\pi^\sharp_s(f \fatsemi g)\right)\\ &= \operatorname{cod}\left(\pi^\sharp_s(f) \fatsemi \pi^\sharp_{\operatorname{cod}\pi^\sharp_s(f)} g\right)\\ &= \operatorname{cod}\left(\pi^\sharp_{\operatorname{cod}\pi^\sharp_s(f)} g\right)\\ &= \alpha_2(\alpha_2(s, f), g).\end{aligned}$$

Going backwards, if we're given a coalgebra $\alpha\colon S \to \mathfrak{c} \triangleleft S$, we obtain a function $\alpha_1\colon S \to \mathfrak{c} \triangleleft 1$ and we define $\mathfrak{s} := \alpha_1^*\mathfrak{c}$ and the cartesian map $\varphi := (\alpha_1, \mathrm{id})\colon \mathfrak{s} \to \mathfrak{c}$ to be the base change from Proposition 4.70. We need to show $\mathfrak{s}$ has a comonoid structure $(\mathfrak{s}, \epsilon, \delta)$ and that φ is a cofunctor. We simply define the counit $\epsilon\colon \mathfrak{s} \to y$ using α_1 and the counit on $\mathfrak{c}$:


$\mathrm{id}_{\alpha_1(s)}$
s
$\mathfrak{s}$
α_1
$\mathrm{id}_{\alpha_1 s}$
$\alpha_1(s)$
$\mathfrak{c}$
$\epsilon_{\mathfrak{c}}$

We comultiplication $\delta\colon \mathfrak{s} \to \mathfrak{s} \triangleleft \mathfrak{s}$ using α_2 for the codomain, and using the composite $\fatsemi$ from $\mathfrak{c}$:



In Exercise 9.10 we check that $(\mathfrak{s}, \epsilon, \delta)$ really is a comonoid, that $(\alpha_1, \mathrm{id})\colon \mathfrak{s} \nrightarrow \mathfrak{c}$ is a cofunctor, that the roundtrips between cartesian cofunctors and coalgebras are identities, and that these assignments are functorial.

$4 \cong 5$**:** This is straightforward and was mentioned in Definition 9.8.

$5 \cong 6$**:** A right $\mathsf{0}$-module is in particular a polynomial $m \in \mathbf{Poly}$ and a map $\rho\colon m \to m \triangleleft \mathsf{0}$ such that $(m \triangleleft \epsilon) \circ \rho = \mathrm{id}_m$. This implies ρ is monic, which itself implies by Proposition 4.17 that m must be constant since $m \triangleleft \mathsf{0}$ is constant. This makes $m \triangleleft \epsilon$ the identity, at which point ρ must also be the identity. Conversely, for any set M, the corresponding constant polynomial is easily seen to make the diagrams in (9.3) commute.

$5 \cong 7$**:** By the adjunction in Proposition 1.22 and the fully faithful inclusion $\mathbf{Set} \to \mathbf{Poly}$ of sets as constant polynmials, Proposition 4.2, we have isomorphisms

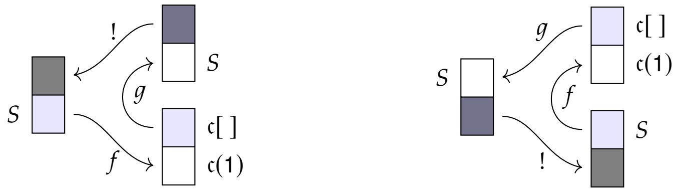
$$\mathbf{Poly}(Sy, \mathfrak{c} \triangleleft Sy) \cong \mathbf{Set}(S, \mathfrak{c} \triangleleft Sy \triangleleft \mathsf{1}) = \mathbf{Set}(S, \mathfrak{c} \triangleleft S) \cong \mathbf{Poly}(S, \mathfrak{c} \triangleleft S).$$

One checks easily that if $Sy \to \mathfrak{c} \triangleleft Sy)$ corresponds to $S \to \mathfrak{c} \triangleleft S$ under the above isomorphism, then one is a left module iff the other is.

$7 \cong 8$**:** By **??** we have a natural isomorphism

$$\mathbf{Poly}(Sy, \mathfrak{c} \triangleleft Sy) \cong \mathbf{Poly}(y^S, y^S \triangleleft \mathfrak{c}).$$

In pictures,



The last claim was proven in Proposition 8.14. □

Exercise 9.10 (Solution here). Complete the proof of Theorem 9.9 ($3 \cong 4$) by proving the following.

1. Show that $(\mathfrak{s}, \epsilon, \delta)$ really is a comonoid.
2. Show that $(\alpha_1, \mathrm{id})\colon \mathfrak{s} \nrightarrow \mathfrak{c}$ is a cofunctor.
3. Show that the roundtrips between cartesian cofunctors and coalgebras are identities.
4. Show that the assignment of a $\mathscr{C}$-coalgebra to a cartesian cofunctor over $\mathcal{C}$ is functorial.
5. Show that the assignment of a cartesian cofunctor over $\mathcal{C}$ to a $\mathscr{C}$-coalgebra is functorial. ◊

Let $\mathcal{C}$ be a category. Under the above correspondence, the terminal functor $\mathcal{C} \to \mathbf{Set}$ corresponds to the identity discrete opfibration $\mathcal{C} \to \mathcal{C}$, the identity cofunctor $\mathscr{C} \to \mathscr{C}$, a certain left $\mathscr{C}$ module with carrier $\mathscr{C}(1)y$ which we call the *canonical left $\mathscr{C}$-module*, a certain constant left $\mathscr{C}$ module with carrier $\mathscr{C}(1)$ which we call the *canonical $(\mathscr{C}, 0)$-bimodule*, and a certain representable right $\mathscr{C}$-module with carrier $y^{\mathscr{C}(1)}$ which we call the *canonical right $\mathcal{C}$-module*.

Exercise 9.11 (Solution here). For any object $c \in \mathcal{C}$, consider the representable functor $\mathcal{C}(c, -)\colon \mathcal{C} \to \mathbf{Set}$. What does it correspond to as a

1. discrete opfibration over $\mathcal{C}$?
2. cartesian cofunctor to $\mathscr{C}$?
3. linear left $\mathscr{C}$-module?
4. constant left $\mathscr{C}$-module?
5. $(\mathscr{C}, 0)$-bimodule?
6. representable right $\mathscr{C}$-module?
7. dynamical system with comonoid interface $\mathscr{C}$? ◊

Exercise 9.12 (Solution here). We saw in Theorem 9.9 that the category ${}_{\mathcal{C}}\mathbf{Mod}_0$ of $(\mathcal{C}, 0)$-bimodule has a very nice structure: it's the topos of copresheaves on $\mathcal{C}$.

1. What is a $(0, \mathcal{C})$-bimodule?
2. What is ${}_0\mathbf{Mod}_{\mathcal{C}}$? ◊

Recall from Proposition 4.70 that for any function $f\colon A \to B$, we have a base-change functor $f^*\colon B\mathbf{Poly} \to A\mathbf{Poly}$ and a cartesian morphism $f^*p \to p$ for any polynomial p and isomorphism $p(1) \cong B$.

Proposition 9.13. Let $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ be a comonoid and suppose that $\rho\colon m \to m \triangleleft \mathfrak{c}$ is a right $\mathscr{C}$-module. Then for any set A and function $f\colon A \to m \triangleleft 1$, the polynomial f^*m has an induced right module structure ρ_f fitting into the commutative square below:

$$\begin{tikzcd}
f^*m \arrow[r, "\rho_f"] \arrow[d] & f^*m \triangleleft \mathfrak{c} \arrow[d] \\
m \arrow[r, "\rho"'] & m \triangleleft \mathfrak{c}
\end{tikzcd}$$

Proof. The pullback diagram to the left defines $f^*(m)$ and that to the right is its composition with $\mathfrak{c}$

$$\begin{tikzcd}
f^*m \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & m \arrow[d] \\
A \arrow[r, "f"'] & m \triangleleft 1
\end{tikzcd}
\qquad
\begin{tikzcd}
f^*m \triangleleft \mathfrak{c} \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & m \triangleleft \mathfrak{c} \arrow[d] \\
A \arrow[r, "f"'] & m \triangleleft 1
\end{tikzcd}$$

which is again a pullback by Theorem 5.52 and Exercise 5.26. Now the map $\rho\colon m \to m \triangleleft \mathfrak{c}$ induces a map $\rho_f\colon f^*(m) \to f^*(m) \triangleleft c$; we claim it is a right module. It suffices to check that ρ_f interacts properly with ϵ and δ, which we leave to Exercise 9.14. □

Exercise 9.14 (Solution here). Let $\mathfrak{c}, \epsilon, \delta)$, $\rho\colon m \to m \triangleleft \mathfrak{c}$, and $f\colon A \to m \triangleleft 1$ be as in Proposition 9.13. Complete the proof of that proposition as follows:

1. Show that $\rho_f \mathbin{⨟} \epsilon = \mathrm{id}_m$
2. Show that $\rho_f \mathbin{⨟} (f^*m \triangleleft \delta) = \rho_f \mathbin{⨟} (\rho_f \triangleleft \mathfrak{c})$. ◊

Definition 9.15 (Polynomial functors of many variables)**.** GK definition.

Proposition 9.16. The bicategory of polynomial functors in the sense of [**GK**] is precisely that of bimodules between discrete categories.

Proof. ** □

Example 9.17 (Cellular automata). In Example 3.67 and Exercise 3.68 we briefly discussed cellular automata; here we will discuss another way that cellular automata show up, this time in terms of bimodules.

Suppose that $\mathrm{src}, \mathrm{tgt}\colon A \rightrightarrows V$ is a graph, and consider the polynomial

$$g \coloneqq \sum_{v \in V} y^{\mathrm{src}^{-1}(v)}$$

so that positions are vertices and directions are emanating arrows. It carries a natural bimodule structure

$$\begin{tikzcd} Vy & Vy \arrow[l, "g"'] \end{tikzcd}$$

where the right structure map uses tgt; see Exercise 9.18 for details. A bimodule

$$\begin{tikzcd} Vy & 0 \arrow[l, "T"'] \end{tikzcd}$$

can be identified with a functor $T\colon V \to \mathbf{Set}$, i.e. it assigns to each vertex $v \in V$ a set. Let's call $T(v)$ the color set at v; for many cellular automata we will put $T(v) \cong 2$ for each v.

Then a cellular automata on g with color sets T is given by a map

$$\begin{tikzcd} Vy & Vy \arrow[l, "g"'] & 0 \arrow[l, "T"'] \arrow[ll, bend left=40, "T", "\Downarrow\alpha"'] \end{tikzcd}$$

Indeed, for every vertex $v \in V$ the map α gives a function

$$\prod_{\mathrm{src}(a)=v} T(\mathrm{tgt}(a)) \xrightarrow{\alpha_v} T(v),$$

which we call the *update* function. In other words, given the current color at the target of each arrow emanating from v, the function α_v returns a new color at v.

Note that if $V \in {}_{Vy}\mathbf{Mod}_0$ is the terminal object, then the composite $Vy \xleftarrow{g}\!\triangleleft\; Vy \xleftarrow{V}\!\triangleleft\; 0$ is again V.

Exercise 9.18 (Solution here). Let src, tgt: $A \rightrightarrows V$ and g and T be as in Example 9.17.

1. Give the structure map $\lambda \colon g \to Vy \triangleleft g$
2. Give the structure map $\rho \colon g \to g \triangleleft Vy$.
3. Give the set T and the structure map $T \to Vy \triangleleft T$ corresponding to the functor $V \to \mathbf{Set}$ that assigns 2 to each vertex. ◊

Example 9.19 (Running a cellular automaton). Let g be a graph on vertex set V, let T assign a color set to each $v \in V$, and let α be the update function for a cellular automaton. As in Example 9.17, this is all given by a diagram

$$\begin{tikzcd}
Vy & Vy \arrow[l, "g"'] & 0 \arrow[l, "T"'] \arrow[ll, bend left, "T"] \arrow[ll, phantom, "\Downarrow\alpha"]
\end{tikzcd}$$

To run the cellular automaton, one simply chooses a starting color in each vertex. We call this an initialization; it is given by a map of bimodules

$$\begin{tikzcd}
Vy & Vy \arrow[l, bend right, "Vy"'] \arrow[l, bend left, "T"] \arrow[l, phantom, "\Downarrow\sigma"]
\end{tikzcd} \tag{9.20}$$

Now to run the cellular automaton on that initialization for $k \in \mathbb{N}$ steps is given by the composite

$$\begin{tikzcd}
Vy \arrow[d, equal] & \cdots \arrow[l, "g"'] & Vy \arrow[l, "g"'] \arrow[d, equal] & Vy \arrow[l, "g"'] \arrow[d, equal] & 0 \arrow[l, "g"'] \arrow[llll, bend right, "V"'] \arrow[d, equal] \\
Vy \arrow[d, equal] & \cdots \arrow[l, "g"'] & Vy \arrow[l, "g"'] \arrow[d, equal] & Vy \arrow[l, "g"'] & 0 \arrow[l, "T"'] \arrow[d, equal] \\
Vy \arrow[d, equal] & \cdots \arrow[l, "g"'] & Vy \arrow[l, "g"'] \arrow[d, Rightarrow, "\alpha\circ\cdots\circ\alpha"] & & 0 \arrow[ll, "T"'] \arrow[d, equal] \\
Vy & & & & 0 \arrow[llll, "T"]
\arrow[from=1-4, to=2-5, phantom, "\Downarrow\sigma"]
\arrow[from=2-4, to=3-4, phantom, "\Downarrow\alpha"]
\end{tikzcd}$$

Exercise 9.21 (Solution here). Explain why (9.20) models an initialization, i.e. a way to choose a starting color in each vertex. ◊

Proposition 9.22. Let $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ be a comonoid in **Poly**. For any set G, the polynomial $y^G \triangleleft \mathfrak{c}$ has a natural right $\mathscr{C}$-module structure.

Proof. We use the map $(y^G \triangleleft \delta)\colon (y^G \triangleleft \mathfrak{c}) \to (y^G \triangleleft \mathfrak{c} \triangleleft \mathfrak{c})$. It satisfies the unitality and associativity laws because $\mathfrak{c}$ does. □

We can think of elements of G as "generators". Then if $i'\colon G \to \mathfrak{c} \triangleleft 1$ assigns to every generator an object of a category $\mathcal{C}$, then we should be able to find the free $\mathcal{C}$-set that i' generates.

Proposition 9.23. Functions $i'\colon G \to \mathfrak{c} \triangleleft 1$ are in bijection with positions $i \in y^G \triangleleft \mathfrak{c} \triangleleft 1$. Let $m := i^*(y^G \triangleleft \mathfrak{c})$ and let ρ_i be the induced right $\mathscr{C}$-module structure from Proposition 9.13. Then ρ_i corresponds to the free $\mathcal{C}$-set generated by i'.

Proof. The polynomial m has the following form:

$$m \cong y^{\sum_{g \in G} \mathfrak{c}[i'(g)]}$$

In particular ρ_i is a representable right $\mathscr{C}$-module, and we can identify it with a $\mathcal{C}$-set by Theorem 9.9. The elements of this $\mathcal{C}$-set are pairs (g, f), where $g \in G$ is a generator and $f\colon i'(g) \to \operatorname{cod}(f)$ is a morphism in $\mathcal{C}$ emanating from $i'(g)$. It is easy to see that the module structure induced by Proposition 9.22 is indeed the free one. □

Exercise 9.24 (Solution here). Let $\mathcal{C}$ be a category and $i \in \mathcal{C}$ an object.

1. Consider i as a map $y \to \mathfrak{c}$. Show that the vertical-cartesian factorization of this map is $y \to y^{\mathfrak{c}[i]} \xrightarrow{\varphi} \mathfrak{c}$.
2. Use Proposition 5.59 to show that $y^{\mathfrak{c}[i]} \triangleleft \mathfrak{c} \to \mathfrak{c} \triangleleft \mathfrak{c}$ is cartesian.
3. Show that there is a commutative square

$$\begin{tikzcd}
y^{\mathfrak{c}[i]} \arrow[r, "\delta^i"] \arrow[d, "\varphi"'] \arrow[dr, phantom, very near start, "\lrcorner"] & y^{\mathfrak{c}[i]} \triangleleft \mathfrak{c} \arrow[d, "\text{cart}"] \\
\mathfrak{c} \arrow[r, "\delta"'] & \mathfrak{c} \triangleleft \mathfrak{c}
\end{tikzcd}$$

4. Show that this square is a pullback, as indicated.
5. Show that δ^i makes $y^{\mathfrak{c}[i]}$ a right $\mathscr{C}$-module. ◊

The map δ^i can be seen as the restriction of $\delta\colon \mathfrak{c} \to \mathfrak{c} \triangleleft \mathfrak{c}$ to a single starting position.

We can extend this to a functor $\mathcal{C} \to {}_y\mathbf{Mod}_{\mathcal{C}}$ that sends the object i to $y^{\mathfrak{c}[i]}$. Given a morphism $f\colon i \to i'$ in $\mathcal{C}$, we get a function $\mathfrak{c}[i'] \to \mathfrak{c}[i]$ given by composition with f, and hence a map of polynomials $y^{\mathfrak{c}[f]}\colon y^{\mathfrak{c}[i]} \to y^{\mathfrak{c}[i']}$.

Exercise 9.25 (Solution here).

1. Show that $y^{\mathfrak{c}[f]}$ is a map of right $\mathscr{C}$-modules.
2. Show that the construction $y^{\mathfrak{c}[f]}$ is functorial in f. ◊

Definition 9.26 (Yoneda)**.** Let $\mathcal{C}$ be a category. We refer to the above functor $y^{\mathfrak{c}[-]} \colon \mathcal{C} \to {}_{y}\mathbf{Mod}_{\mathcal{C}}$ as the *Yoneda* functor.

Proposition 9.27. The Yoneda functor $y^{\mathfrak{c}[-]} \colon \mathcal{C} \to {}_{y}\mathbf{Mod}_{\mathcal{C}}$ is fully faithful.

Proof. ** □

Proposition 9.28. For any functor $F \colon \mathcal{C} \to \mathbf{Poly}$, the limit polynomial $\lim_{c \in \mathcal{C}} F(c)$ is obtained by composing with the canonical right bimodule $y^{\mathrm{Ob}(\mathcal{C})}$

$$\begin{tikzcd}
y \arrow[r, "F"] \arrow[rr, bend right, "\lim F"'] & \mathscr{C} \arrow[r, "y^{\mathrm{Ob}(\mathcal{C})}"] & y
\end{tikzcd}$$

Proposition 9.29. Let $\mathscr{C}$ be a comonoid. For any set I and right $\mathscr{C}$-modules $(m_i)_{i \in I}$, the coproduct $m := \sum_{i \in I} m_i$ has a natural right-module structure. Moreover, each representable summand in the carrier m of a right $\mathscr{C}$-module is itself a right-$\mathscr{C}$ module and m is their sum.

Proof. ** □

Proposition 9.30. If $m \in \mathbf{Poly}$ is equipped with both a right $\mathscr{C}$-module and a right $\mathscr{D}$-module structure, we can naturally equip m with a $(\mathscr{C} \times \mathscr{D})$-module structure.

Proof. It suffices by Proposition 9.29 to assume that $m = y^M$ is representable. But a right $\mathscr{C}$-module with carrier y^M can be identified with a cofunctor $My^M \to \mathscr{C}$.

Thus if y^M is both a right-$\mathscr{C}$ module and a right-$\mathscr{D}$ module, then we have comonoid morphisms $\mathscr{C} \leftarrow My^M \to \mathscr{D}$. This induces a unique comonoid morphism $My^M \to (\mathscr{C} \times \mathscr{D})$ to the product, and we identify it with a right-$(\mathscr{C} \times \mathscr{D})$ module on y^M. □

9.1 Morphisms between bimodules

Proposition 9.31. For any two categories $\mathfrak{c}, \mathfrak{d}$, the category ${}_{\mathfrak{d}}\mathbf{Mod}_{\mathfrak{c}}$ of bimodules between them is a rig category: it has a terminal object (carried by $\mathfrak{d} \triangleleft 1$), an initial object (carried by 0), as well as products that distribute over coproducts.

Proof. For bimodules $\mathfrak{d} \triangleleft\!\xleftarrow{m}\!\triangleleft\ \mathfrak{c}$ and $\mathfrak{d} \triangleleft\!\xleftarrow{n}\!\triangleleft\ \mathfrak{c}$, the coproduct has carrier $m + n$, and the product has carrier $m \times_{d \triangleleft 1} n$. **Finish** □

Proposition 9.32. For any two categories $\mathfrak{c}, \mathfrak{d}$, the category ${}_{\mathfrak{d}}\mathbf{Mod}_{\mathfrak{c}}$ has all limits and is extensive.

Proof. ** □

9.1.1 Adjoint bimodules

Exercise 9.33 (Solution here). Recall that there is a unique $(0, \mathcal{C})$-bimodule, namely $0 \triangleleft\!\xleftarrow{0}\!\triangleleft\ \mathcal{C}$.

1. Show that 0 has a left adjoint $\mathcal{C} \triangleleft\!\longleftarrow\!\triangleleft\ 0$; what is its carrier?
2. Show that 0 has a right adjoint $\mathcal{C} \triangleleft\!\longleftarrow\!\triangleleft\ 0$; what is its carrier? ◊

9.2 Bimodules as data migration functors

Proposition 9.34. Let $\mathcal{C}$ and $\mathcal{D}$ be categories; the following conditions on a functor $F \colon \mathcal{C}\text{-}\mathbf{Set} \to \mathcal{D}\text{-}\mathbf{Set}$ are equivalent.

1. F is composition with a $(\mathcal{D}, \mathcal{C})$-bimodule.
2. F is a parametric right adjoint in the sense of [].
3. F ... $\Sigma\Pi\Delta$
4. F profunctor + discrete opfibration
5. F preserves connected limits.

Warning 9.35. **Cat** is not locally cartesian closed, so our notion of $\Sigma\Pi\Delta$ is not the one in **[gambino2013polynomial]**.

Consider a category $\mathcal{C}$ and a bimodule $\mathcal{C} \triangleleft\!\xleftarrow{p}\!\triangleleft\ \mathcal{C}$ from it to itself. Since $\mathcal{C}$-**Set** is locally cartesian closed, one could ask whether p arises from the bridge diagram

$$\begin{tikzcd}
& E \arrow[r] \arrow[dl] & B \arrow[dr] & \\
1 & & & 1
\end{tikzcd}$$

for a map of copresheaves $E \to B$. It may not!

As a counterexample, consider the walking arrow category $\boxed{\bullet \to \bullet}$. The functor sending the copresheaf $A \to B$ to $\emptyset \to A$ is a prafunctor, but it is not representable by a map of copresheaves; see Exercise 9.36

Exercise 9.36 (Solution here). Prove the counterexample from Warning 9.35. ◊

Definition 9.37. Pra-functors Let $\mathcal{C}$ and $\mathcal{D}$ be categories. A *pra-functor* (also called a *parametric right adjoint functor*) $\mathcal{C}$-**Set** → $\mathcal{D}$-**Set** is one satisfying any of the conditions of Proposition 9.34

When a polynomial

$$m := \sum_{i \in m(1)} y^{m[i]}$$

is given the structure of a $(\mathcal{D}, \mathcal{C})$-bimodule, the symbols in that formula are given a hidden special meaning:

$$m(1) \in \mathcal{D}\text{-}\mathbf{Set} \qquad \text{and} \qquad m[i] \in \mathcal{C}\text{-}\mathbf{Set}$$

Thus $m(1)$ is a database instance on $\mathcal{D}$; in particular, each position in $i \in m(1)$ is a row in that instance. And each $m[i]$ is a database instance on $\mathcal{C}$; in particular, each direction $d \in m[i]$ is a row in that instance.

Before we knew about bimodule structures, what we called positions and directions—and what we often think of as outputs and inputs of a system—were understood as each forming an ordinary set. In the presence of a bimodule structure, the positions $m(1)$ have been organized into a $\mathcal{D}$-set and the directions $m[i]$ have been organized into a $\mathcal{C}$-set for each position i. We are listening for $\mathcal{C}$-sets and positioning ourselves in a $\mathcal{D}$-set.

Theorem 9.38. The 2-functor Copsh: **Mod** → **Cat** given by $\mathcal{C} \mapsto {}_{\mathcal{C}}\mathbf{Mod}_0$ is fully faithful.

Proof. The functor Copsh sends $\mathcal{C}$ to the corresponding copresheaf category $\mathcal{C}$-**Set** by Theorem 9.9.

Finish □

Exercise 9.39 (Solution here). Consider the query on $y^{\mathbb{N}}$ from **??** asking for siblings.

1. Write out the corresponding $y^{\mathbb{N}}$ instance in table form.
2. Draw it as a dynamical system.

◊

9.3 Monoidal operations

Theorem 9.40. For any categories $\mathcal{C}$ and $\mathcal{D}$, the functor

$$_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}} \xrightarrow{\cong} \mathbf{Cat}(\mathcal{C},\ {}_{y}\mathbf{Mod}_{\mathcal{D}})$$

is an equivalence.

Proof. □

Corollary 9.41. For any categories $\mathcal{C}_1, \mathcal{C}_2$, and $\mathcal{D}$, the functor

$$_{\mathcal{C}_1\otimes\mathcal{C}_2}\mathbf{Mod}_{\mathcal{D}} \to \mathbf{Cat}(\mathcal{C}_1,\ {}_{\mathcal{C}_2}\mathbf{Mod}_{\mathcal{D}})$$

is an equivalence.

Proof. ** $\mathcal{C}_1 \otimes \mathcal{C}_2$ is the usual product of categories ** □

Corollary 9.42. Let $\mathscr{C}$ be a comonoid. The category of left $\mathscr{C}$ modules is equivalent to the category of functors $\mathcal{C} \to \mathbf{Poly}$.

Proof. Use Theorem 9.40 with $\mathcal{D} = y$ and the equivalence ${}_{y}\mathbf{Mod}_{y} \cong \mathbf{Poly}$. □

Proposition 9.43. The monoidal operation + is a coproduct in **Mod**. That is, for any categories $\mathcal{C}, \mathcal{D}, \mathcal{E}$ there is an equivalence of categories

$$_{\mathcal{C}+\mathcal{D}}\mathbf{Mod}_{\mathcal{E}} \cong\ {}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{E}} \times {}_{\mathcal{D}}\mathbf{Mod}_{\mathcal{E}}$$

Proof. This follows from Corollary 9.41:

$$\begin{aligned}
{}_{\mathcal{C}+\mathcal{D}}\mathbf{Mod}_{\mathcal{E}} &\cong \mathbf{Cat}(\mathcal{C}+\mathcal{D},\ {}_{y}\mathbf{Mod}_{\mathcal{E}})\\
&\cong \mathbf{Cat}(\mathcal{C},\ {}_{y}\mathbf{Mod}_{\mathcal{E}}) \times \mathbf{Cat}(\mathcal{D},\ {}_{y}\mathbf{Mod}_{\mathcal{E}})\\
&\cong\ {}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{E}} \times {}_{\mathcal{D}}\mathbf{Mod}_{\mathcal{E}}\square
\end{aligned}$$

In fact, + is almost a biproduct in **Mod**, except the required equivalence is replaced by an adjunction.

Proposition 9.44. For any categories $\mathcal{C}, \mathcal{D}, \mathcal{E}$ there is an adjunction

$$\begin{tikzcd}
{}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}+\mathcal{E}} \arrow[r, shift left=2] \arrow[r, phantom, "\Rightarrow"] & {}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}} \times {}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{E}} \arrow[l, shift left=2]
\end{tikzcd}$$

Proof. By Corollary 9.41, it suffices to show that there is an adjunction

$$\begin{tikzcd}
{}_{y}\mathbf{Mod}_{\mathcal{D}+\mathcal{E}} \arrow[r, shift left=2] \arrow[r, phantom, "\Rightarrow"] & {}_{y}\mathbf{Mod}_{\mathcal{D}} \times {}_{y}\mathbf{Mod}_{\mathcal{E}} \arrow[l, shift left=2]
\end{tikzcd}\ .$$

** □

Proposition 9.45. If $\mathfrak{c} \overset{m_1}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} \mathfrak{d}$ and $\mathfrak{c} \overset{m_2}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} \mathfrak{d}$ are bimodules, then so are

$$\mathfrak{c} \overset{m_1+m_2}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} \mathfrak{d} \qquad \text{and} \qquad \mathfrak{c} \overset{m_1\times_{\mathfrak{c}(1)}m_2}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} \mathfrak{d}$$

Proposition 9.46. If $\mathfrak{c}_1 \overset{m_1}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} \mathfrak{d}_1$ and $\mathfrak{c}_2 \overset{m_2}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} \mathfrak{d}_2$ are bimodules, then so are

$$\mathfrak{c}_1 + \mathfrak{c}_2 \overset{m_1+m_2}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} \mathfrak{d}_1 + \mathfrak{d}_2 \qquad \text{and} \qquad \mathfrak{c}_1 \otimes \mathfrak{c}_2 \overset{m_1\otimes m_2}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} \mathfrak{d}_1 \otimes \mathfrak{d}_2$$

9.4 Composing bimodules

We denote the composite of $\mathfrak{e} \overset{n}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} \mathfrak{d} \overset{m}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} \mathfrak{c}$ by

$$\mathfrak{e} \overset{n \triangleleft_{\mathfrak{d}} m}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} \mathfrak{c}.$$

Recall the Yoneda functor $y^{\mathfrak{c}[-]}\colon \mathcal{C} \to {}_y\mathbf{Mod}_{\mathcal{C}}$ from Definition 9.26. Using bimodule composition, we obtain the composite functor

$$\mathcal{C} \times {}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}} \xrightarrow{y^{\mathfrak{c}[-]} \times {}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}}} {}_y\mathbf{Mod}_{\mathcal{C}} \times {}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}} \overset{[}{\longrightarrow} \triangleleft_{\mathcal{C}}]\, {}_y\mathbf{Mod}_{\mathcal{D}}.$$

By the cartesian closure of **Cat**, this can be identified with a functor ${}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}} \to \mathbf{Cat}(\mathcal{C}, {}_y\mathbf{Mod}_{\mathcal{D}})$.

Definition 9.47. For any category $\mathcal{C}$, define the category of *polynomials in* $\mathcal{C}$, denoted $\mathbf{Set}[\mathcal{C}]$, by

$$\mathbf{Set}[\mathcal{C}] := {}_y\mathbf{Mod}_{\mathcal{C}}$$

Example 9.48. When $\mathcal{C} = y$, we have $\mathbf{Set}[y] \cong \mathbf{Poly}$, and for any set I considered as a discrete category Iy, we have $\mathbf{Set}[Iy]$ is the category of polynomial functors in I many variables.

For arbitrary $\mathcal{C}$, we can think of $\mathbf{Set}[\mathcal{C}]$ as a polynomial rig with variables in $\operatorname{Ob}\mathcal{C}$, but with arbitrary limits replacing the mere products you would find when $\mathcal{C}$ is a discrete category.

Proposition 9.49. For any category $\mathcal{C}$, the category $\mathbf{Set}[\mathcal{C}]$ is the free coproduct completion of $(\mathcal{C}\text{-}\mathbf{Set})^{\mathrm{op}}$.

Proof. ** □

Proposition 9.50 (Carrier functor)**.** For any category $\mathcal{C}$, the carrier functor

$$\mathbf{Set}[\mathcal{C}] \to \mathbf{Poly}$$

sending $m \to m \triangleleft \mathfrak{c}$ to $m \in \mathbf{Poly} \cong \mathbf{Set}[y]$ is given by composition with the principle $\mathcal{C}$-module $\mathcal{C} \overset{\check{\mathfrak{c}}}{\triangleleft\!\!\!-\!\!\!-\!\!\!\triangleleft} y$.

Proof. ** □

Proposition 9.51. The functor $\eta_{\mathcal{C}} \colon \mathcal{C} \to \mathbf{Set}[\mathcal{C}]$ corresponding to the unit bimodule $\mathfrak{c} \overset{\mathfrak{c}}{\triangleleft\!\!\!-\!\!\!-\!\!\!\triangleleft} \mathfrak{c}$ sends each object $i \in \mathfrak{c}(1)$ to the bimodule $y \overset{y^{\mathfrak{c}[i]}}{\triangleleft\!\!\!-\!\!\!-\!\!\!\triangleleft} \mathfrak{c}$.

Proposition 9.52. The functor $(\mathcal{C}\text{-}\mathbf{Set})^{\mathrm{op}} \to \mathbf{Set}[\mathcal{C}]$ has a left adjoint,

$$\begin{tikzcd}
(\mathcal{C}\text{-}\mathbf{Set})^{\mathrm{op}} \arrow[r, shift left=2, "y^{-}"] \arrow[r, phantom, "\Leftarrow"] & \mathbf{Set}[\mathcal{C}] \arrow[l, shift left=2, "\Gamma"]
\end{tikzcd} .$$

Proof. ** □

Theorem 9.53. For any categories $\mathcal{C}, \mathcal{D}$, there is an adjunction

$$\begin{tikzcd}
({}_{\mathcal{C}^{\mathrm{op}} \times \mathcal{D}}\mathbf{Mod}_0)^{\mathrm{op}} \arrow[r, shift left=2] \arrow[r, phantom, "\Rightarrow"] & {}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}} \arrow[l, shift left=2]
\end{tikzcd}$$

such that the left adjoint is a fully faithful inclusion of profunctors into bimodules.

Proposition 9.54. For any category $\mathcal{C}$, the carrier functor creates limits.

Proof. ** □

Theorem 9.55. For any $(\mathcal{C}, \mathcal{D})$-bimodule m, the functor $\mathbf{Set}[\mathcal{C}] \to \mathbf{Set}[\mathcal{D}]$ defined by composition with m preserves constants, coproducts, and all small limits; in particular, it's a map of rig categories.

Proof. ** □

Proposition 9.56. The composite of a linear left $\mathcal{C}$-module and a representable right $\mathcal{C}$-module is the set of natural transformations between the corresponding copresheaves.

Proposition 9.57. For any categories $\mathcal{C}$ and $\mathcal{D}$ and $(\mathcal{D}, \mathcal{C})$-bimodule $\mathfrak{d} \overset{m}{\triangleleft\!\!\longleftarrow\!\!\triangleleft} \mathfrak{c}$, we have

$$m \triangleleft_{\mathfrak{c}} \mathfrak{c} \cong m$$

where $\mathfrak{c} \overset{\mathfrak{c}}{\triangleleft\!\!\longleftarrow\!\!\triangleleft} y$ is the regular left $\mathfrak{c}$-module.

Theorem 9.58. For any category $\mathfrak{c}$, there is an adjunction

$$\begin{tikzcd}
\mathbf{Cat}^\sharp/\mathfrak{c} \arrow[r, shift left=2] \arrow[r, phantom, "\Rightarrow"] & {}_{\mathfrak{c}}\mathbf{Mod}_0 \arrow[l, shift left=2, "\int"]
\end{tikzcd}$$

where $\int$ is the usual category-of-elements ("Grothendieck") construction.

Moreover, $\int$ is fully faithful, and the unit map on $\mathbf{Cat}^\sharp/c$ sends $d \nrightarrow c$ to its vertical-cartesian factorization; see Proposition 6.98.

9.5 Exercise solutions

Chapter 10

The framed bicategory

A type of categorical structure called framed bicategory—also known as a proarrow equipment—gives a nice way to organize what's going on with comonoids, cofunctors, bimodules, and maps between them. A framed bicategory is the type of double category, so it has objects, vertical morphisms, horizontal morphisms, and 2-cells. Thus we will discuss a framed bicategory $\mathbb{P}$ for which the objects are categories, the vertical morphisms are cofunctors, and the horizontal morphisms are parametric right adjoints.

10.1 Adjoint bimodules

Proposition 10.1. A functor $F \colon \mathcal{C} \to \mathcal{D}$ gives rise to a right adjoint bimodule (= left adjoint pra) $\mathfrak{c} \xleftarrow{\Delta_F} \mathfrak{d}$.

Proof. The carriers of $\mathfrak{c} \xleftarrow{\Delta_F} \mathfrak{d}$ and its right adjoint $\mathfrak{d} \xleftarrow{\Pi_F} \mathfrak{c}$ are the polynomials

$$\sum_{i \in \mathfrak{c}(1)} y^{Fi} \qquad \text{and} \qquad \sum_{j \in \mathfrak{d}(1)} y^{\sum_{j \to j'}(Fi = j')}$$

respectively. One may recognize the exponent of the latter, namely $\sum_{j \to j'}(Fi = j')$ as the set of objects in the comma category $(j \downarrow F)$, which shows up in the usual conical limit formula for the right Kan extension Π_F. □

Proposition 10.2. A cofunctor $\varphi \colon \mathcal{C} \nrightarrow \mathcal{D}$ gives rise to a left adjoint bimodule (= right adjoint pra) $\mathfrak{c} \xleftarrow{\widehat{\varphi}} \mathfrak{d}$; we denote its right adjoint bimodule (= left adjoint pra) by $\check{\varphi}$,

$$\begin{tikzcd} \mathfrak{c} \arrow[r, bend left, "\check{\varphi}"] \arrow[r, phantom, "\Rightarrow"] & \mathfrak{d} \arrow[l, bend left, "\widehat{\varphi}"] \end{tikzcd} \tag{10.3}$$

Note that the adjunction notation in (10.3) is unambiguous: if you read $\mathfrak{c} \triangleleft\!\!\relbar\!\!\relbar\!\!\triangleleft\ \mathfrak{d}$ in the bimodule direction—i.e. as a map from $\mathfrak{c}$ to $\mathfrak{d}$—then the notation indicates that $\widehat{\varphi}$ is the left adjoint. If instead you read $\mathfrak{c} \triangleleft\!\!\relbar\!\!\relbar\!\!\triangleleft\ \mathfrak{d}$ in the pra direction—i.e. as a map from $\mathfrak{d}$ to $\mathfrak{c}$—then the notation indicates that $\widehat{\varphi}$ is the right adjoint.

Proof. ** □

Proposition 10.4. For any category $\mathcal{C}$, the pras associated to the terminal cofunctor $\mathcal{C} \nrightarrow y$ are the principal left and right $\mathscr{C}$-modules.

Example 10.5 (Types on database tables). For a category $\mathcal{C}$, a functor $\mathcal{C} \to \mathbf{Set}$ doesn't appear to have actual attributes, e.g. string, integers, etc., attached to its elements. To correct this, let's add a type T_c to each table in c, and ask that each row in c is assigned an element of T_c.

To do this, we give a functor $T\colon \text{Ob}(\mathcal{C}) \to \mathbf{Set}$, i.e. $\text{Ob}(\mathcal{C}) \overset{T}{\triangleleft\!\!\relbar\!\!\relbar\!\!\triangleleft}\ 0$. We also have the canonical cofunctor $o\colon \mathcal{C} \nrightarrow \text{Ob}(\mathcal{C})$. Now given an arbitrary instance $\mathcal{C} \overset{I}{\triangleleft\!\!\relbar\!\!\relbar\!\!\triangleleft}\ 0$, there are two things we could do. We could push it forward along the left adjoint prafunctor $\text{Ob}(\mathcal{C}) \triangleleft\!\!\relbar\!\!\relbar\!\!\triangleleft\ \mathcal{C}$ and then map it to T.

Perhaps better would be to compose T with the right adjoint prafunctor $\mathcal{C} \triangleleft\!\!\relbar\!\!\relbar\!\!\triangleleft\ \ \text{Ob}(\mathcal{C}) \overset{T}{\triangleleft\!\!\relbar\!\!\relbar\!\!\triangleleft}\ 0$ and map I into that composite. The reason it is better is that the category of coalgebras I equipped with a map into a fixed coalgebra X (e.g. $X = \check{o} \triangleleft_{\text{Ob}\,\mathcal{C}} T$) is equivalent to the category of coalgebras (instances) on the category of elements $\int^{\mathcal{C}} X$. So we have found that instances, even equipped with types, can be understood just in terms of $\mathcal{C}$-sets, beefing up $\mathcal{C}$ if necessary.

Proposition 10.6. A pra $\mathcal{C}\mathbf{Set} \to \mathcal{D}\mathbf{Set}$ is a left adjoint if it is of the form $\Sigma_G \circ \Delta_F$ for functors $\mathcal{C} \xleftarrow{F} \mathcal{X} \xrightarrow{G} \mathcal{D}$, where G is a discrete opfibration.

Proof. ** □

Corollary 10.7. If $\varphi\colon \mathcal{C} \nrightarrow \mathcal{D}$ is a cartesian cofunctor corresponding to a discrete opfibration $F\colon \mathcal{C} \to \mathcal{D}$, then there is an isomorphism $\widehat{\varphi} \cong \Delta_F$.

In particular, $\widehat{\varphi}$ has an extra adjoint

$$\begin{tikzcd}
\mathfrak{c} \arrow[r, bend left=60, "\check{\varphi}"] \arrow[r, bend right=60, "\tilde{\varphi}"'] \arrow[r, phantom, shift left=2.5ex, "\Rightarrow"] \arrow[r, phantom, shift right=2.5ex, "\Leftarrow"] & \mathfrak{d} \arrow[l, "\widehat{\varphi}" description]
\end{tikzcd}$$

Proof. ** □

Proposition 10.8. Every bimodule $\mathfrak{c} \overset{S}{\lhd\!\!-\!\!\!-\!\!\lhd} 0$, is isomorphic to the composite

$$\mathfrak{c} \overset{Sy}{\lhd\!\!-\!\!\!-\!\!\lhd} y \overset{1}{\lhd\!\!-\!\!\!-\!\!\lhd} 0$$

for a bimodule $\mathfrak{c} \overset{Sy}{\lhd\!\!-\!\!\!-\!\!\lhd} y$ with carrier $Sy \in \mathbf{Poly}$.

Moreover, Sy is adjoint to a bimodule $y \overset{y^S}{\lhd\!\!-\!\!\!-\!\!\lhd} \mathfrak{c}$ with carrier $y^S \in \mathbf{Poly}$.

Proof. **Construct this using Theorem 9.58 and Corollary 10.7.** □

Now that we have the double category $\mathbb{P}$, we can consider the squares between cofunctors

$$\begin{tikzcd}
\mathfrak{c} \arrow[r, "\mathfrak{c}"] \arrow[d, "f"'] \arrow[dr, phantom, "\Downarrow"] & \mathfrak{c} \arrow[d, "g"] \\
\mathfrak{d} \arrow[r, "\mathfrak{d}"'] & \mathfrak{d}
\end{tikzcd}$$

Definition 10.9 (Natural co-transformations)**.** Let $\mathcal{C}$ and $\mathcal{D}$ be categories, let $\varphi, \varphi' \colon \mathcal{C} \nrightarrow \mathcal{D}$ be cofunctors. A *natural cotransformation* $\alpha \colon \varphi \to \varphi'$ between them consists of

1. a function $\alpha_1 \colon \operatorname{Ob}(\mathcal{C}) \to \operatorname{Ob}(\mathcal{C})$
2. for each $i \in \operatorname{Ob}(\mathcal{C})$, a morphism $\alpha_i^\sharp \colon i \to \alpha_1(i)$

satisfying two conditions for every $i \in \operatorname{Ob}(\mathcal{C})$. Namely, letting $i' := \alpha_1(i)$ we require:

1. $\varphi_1(i) = \varphi_1'(i')$, call it j, and
2. for each $f \in \mathcal{D}[j]$ we have $\alpha_1(\operatorname{cod} \varphi_i^\sharp(f)) = \operatorname{cod}(\varphi')_{i'}^\sharp(f)$ and the diagram below commutes in $\mathcal{C}$:

$$\begin{tikzcd}
i \arrow[r, "\alpha_i^\sharp"] \arrow[d, "\varphi_i^\sharp(f)"'] & i' \arrow[d, "(\varphi')_{i'}^\sharp(f)"] \\
\operatorname{cod} \varphi_i^\sharp(f) \arrow[r, "\alpha^\sharp_{\operatorname{cod} \varphi_i^\sharp(f)}"'] & \operatorname{cod}(\varphi')_{i'}^\sharp(f)
\end{tikzcd}$$

10.2 Monoidal structures on $\mathbb{P}$

For any bimodules $\mathfrak{c}_1 \overset{m_1}{\lhd\!\!-\!\!\!-\!\!\lhd} \mathfrak{d}_1$ and $\mathfrak{c}_\mathfrak{2} \overset{m_2}{\lhd\!\!-\!\!\!-\!\!\lhd} \mathfrak{d}_\mathfrak{2}$, we can construct a new bimodule in two ways:

$$\mathfrak{c}_1 + \mathfrak{c}_2 \overset{m_1+m_2}{\lhd\!\!-\!\!\!-\!\!\lhd} \mathfrak{d}_1 + \mathfrak{d}_2 \qquad \text{and} \qquad \mathfrak{c}_1 \otimes \mathfrak{c}_\mathfrak{2} \overset{m_1 \otimes m_2}{\lhd\!\!-\!\!\!-\!\!\lhd} \mathfrak{d}_1 \otimes \mathfrak{d}_2$$

One can think of this quite easily in terms of prafunctors: given prafunctors $\mathcal{D}_1$-**Set** → $\mathcal{C}_1$-**Set** and $\mathcal{D}_2$-**Set** → $\mathcal{C}_2$-**Set**, one obtains prafunctors between the coproduct categories and the product categories

$$(\mathcal{D}_1 + \mathcal{D}_2)\text{-}\mathbf{Set} \to (\mathcal{C}_1 + \mathcal{C}_2)\text{-}\mathbf{Set}$$

In this section we will explain how this looks in terms of bimodules, e.g. the respective bimodule structures on $m_1 + m_2$ and $m_1 \otimes m_2$, and show that each of these constructions satisfies all the conditions to be a monoidal structure on the framed bicategory $\mathbb{P}$.

10.3 Discussion and open questions

In this section, we lay out some questions that whose answers may or may not be known, but which were not known to us at the time of writing. They vary from concrete to open-ended, they are not organized in any particular way, and are in no sense complete. Still we hope they may be useful to some readers.

1. What can you say about comonoids in the category of all functors **Set** → **Set**, e.g. ones that aren't polynomial.
2. What can you say about the internal logic for the topos $\mathscr{T}_p$-**Set** of dynamical systems with interface p, in terms of p?
3. How does the logic of the topos $\mathscr{T}_p$ help us talk about issues that might be useful in studying dynamical systems?
4. Morphisms $p \to q$ in **Poly** give rise to left adjoints $\mathscr{T}_p \to \mathscr{T}_q$ that preserve connected limits. These are not geometric morphisms in general; in some sense they are worse and in some sense they are better. They are worse in that they do not preserve the terminal object, but they are better in that they preserve every connected limit not just finite ones. How do these left adjoints translate statements from the internal language of p to that of q?
5. Consider the $\times$-monoids and $\otimes$-monoids in three categories: **Poly**, $\mathbf{Cat}^\sharp$, and $\mathbb{M}\mathbf{od}$. Find examples of these comonoids, and perhaps characterize them or create a theory of them.
6. Is there a functor **Poly** has pullbacks, so one can consider the bicategory of spans in **Poly**. Is there a functor from that to $\mathbb{M}\mathbf{od}$ that sends $p \mapsto \mathscr{T}_p$?
7. Databases are static things, whereas dynamical systems are dynamic; yet we see them both in terms of **Poly**. How do they interact? Can a dynamical system read from or write to a database in any sense?
8. Can we do database aggregation in a nice dynamic way?
9. In the theory of polynomial functors, sums of representable functors **Set** → **Set**, what happens if we replace sets with homotopy types: how much goes through? Is anything improved?
10. Are there any functors **Set** → **Set** that aren't polynomial, but which admit a comonoid structure with respect to composition $(y, \triangleleft)$?
11. Characterize the monads in poly? They're generalizations of one-object operads (which are the Cartesian ones), but how can we think about them?
12. Both functors and cofunctors give left adjoint bimodules: for functors $F \colon \mathscr{D} \to \mathscr{C}$ we use the pullback Δ_F and for cofunctors $G \colon \mathscr{C} \nrightarrow \mathscr{D}$ we use the companion as in **??**. Can we characterize left adjoint bimodules in general?

13. What limits exist in $\mathbf{Cat}^\sharp$? Describe them combinatorially.
14. Since the forgetful functor $U\colon \mathbf{Cat}^\sharp \to \mathbf{Poly}$ is faithful, it reflects monomorphisms: if $f\colon \mathcal{C} \nrightarrow \mathcal{D}$ is a cofunctor whose underlying map on emanation polynomials is monic, then it is monic. Are all monomorphisms in $\mathbf{Cat}^\sharp$ of this form?
15. At first blush it appears that **Poly** may be suitable as the semantics of a language for protocols. Develop such a language or showcase the limitations that make it impossible or inconvenient.
16. Are there polynomials p such that one use something like Gödel numbers to encode logical propositions from the topos tree_p-**Set** into a "language" that p-dynamical systems can "work with"?

10.4 Exercise solutions

Bibliography

[GK12] Nicola Gambino and Joachim Kock. “Polynomial functors and polynomial monads”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 154.1 (Sept. 2012), pp. 153–192 (cit. on p. 174).

[Hed18] Jules Hedges. *Limits of bimorphic lenses*. 2018. eprint: `arXiv:1808.05545` (cit. on p. 39).

[Kel74] G Max Kelly. “Doctrinal adjunction”. In: *Category seminar*. Springer. 1974, pp. 257–280 (cit. on p. 235).

[Lac10] Stephen Lack. “Note on the construction of free monoids”. In: *Applied categorical structures* 18.1 (2010), pp. 17–29 (cit. on p. 231).

[MM92] Saunders MacLane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Springer, 1992 (cit. on p. 28).

[nLa18] Contributors To nLab. *Created limit — nLab*. 2018. URL: `https://ncatlab.org/nlab/show/created+limit` (cit. on p. 214).

[Par69] Robert Paré. “Absolute coequalizers”. In: *Category Theory, Homology Theory and their Applications I*. Ed. by Peter J. Hilton. Berlin, Heidelberg: Springer, 1969, pp. 132–145. ISBN: 978-3-540-36095-7 (cit. on p. 214).

[Por19] Hans-E Porst. “Colimits of monoids”. In: *Theory and Applications of Categories* 34.17 (2019), pp. 456–467 (cit. on p. 214).

[Shu08] Michael Shulman. “Framed bicategories and monoidal fibrations”. In: *Theory and Applications of Categories* 20 (2008), Paper No. 18, 650–738 (cit. on pp. 14, 138, 141).