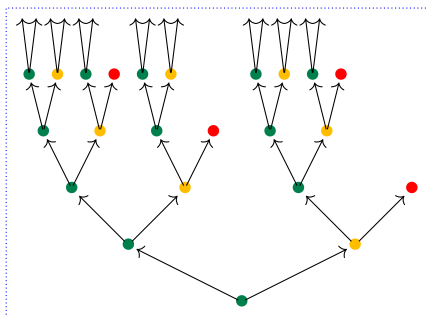
Polynomial Functors: A Mathematical Theory of Interaction



Nelson Niu David I. Spivak

Last updated: July 17, 2024
Source: https://github.com/ToposInstitute/poly

This page intentionally left blank.

Nelson Niu
University of Washington
Seattle, WA

David I. Spivak
Topos Institute
Berkeley, CA

To André Joyal

—D.S.

To my graduate cohort at UW

—N.N.

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*

> And that is the way to get the greatest possible variety, but with all the order there could be; i.e. it is the way to get as much perfection as there could be.
>
> —Gottfried Wilhelm Leibniz
> *Monadology*, 58.

During the Fifth International Conference on Applied Category Theory in 2022, at least twelve of the fifty-nine presentations and two of the ten posters referenced the category of polynomial functors and dependent lenses or its close cousins (categories of optics and Dialectica categories) and the way they model diverse forms of interactive behavior. At the same time, all that is needed to grasp the construction of this category—called **Poly** for short—is an understanding of mathematical sets and functions. There is no need for the theory and applications of polynomial functors to remain the stuff of technical papers; **Poly** is far too versatile, too full of potential, to be kept out of reach.

Informally, a polynomial functor is a collection of elements we call *positions* and, for each position, a collection of elements we call *directions*. There is then a natural notion of a morphism between polynomial functors that sends positions forward and directions backward, modeling two-way communication. From these basic components, category theory allows us to construct an immense array of mathematical gadgets that model a diverse range of interactive processes. In this book, we will establish the theory of polynomial functors and categorical constructions on them while exploring how they model interaction.

Purpose and prerequisites

A categorical theory of general interaction must be interdisciplinary by its very nature. Already, drafts of this text have been read by everyone from algebraic geometers to neuroscientists and AI developers. We hope to extend our reach ever further, to bring together thinkers and tinkerers from a diverse array of backgrounds under a common language by which to study interactive systems categorically. In short—we know about **Poly**; you know about other things; but only our collective knowledge can reveal how **Poly** could be applied to those other things.

As such, we have strived to write a friendly and accessible expository text that can serve as a stepping stone toward further investigations into polynomial functors. We include exercises and, crucially, solutions to guide the learning process; we draw extensive analogies to provide motivation and develop intuition; we pose examples whenever necessary. Proofs may bear far more detail than you would find in a research paper, but not so much detail that it would clutter the key ideas. A few critical proofs are even argued through pictures, yet we contend that they are no less rigorous than the clouds of notation whose places they take.

On the other hand, there is some deep mathematical substance to the work we will discuss, drawing from the well-established theory of categories. Although you will find, for example, a complete proof of the Yoneda lemma within these pages, we don't intend to build up everything from scratch. There are plenty of excellent resources for learning category theory out there, catering to a variety of needs, without adding our own to the mix when our primary goal is to introduce **Poly**. So for the sake of contributing only what is genuinely helpful, we assume a certain level of mathematical background. You are ready to read this book if you can define the following fundamental concepts from category theory, and give examples of each:

- categories,
- functors,
- natural transformations,
- (co)limits,
- adjunctions, and
- (symmetric) monoidal categories.

We will additionally assume a passing familiarity with the language of graph-theoretic trees (e.g. vertices, roots, leaves, paths).

That said, with a little investment on your part, you could very well use this book as a way to teach yourself some category theory. If you have ever tried to learn category theory, only to become lost in abstraction or otherwise overwhelmed by seemingly endless lists of examples from foreign fields, perhaps you will benefit from a focused case study of one particularly fruitful category. If you encounter terms or ideas that you would like to learn more about, we encourage you to look them up elsewhere, and you may find yourself spending a pleasant afternoon doing a deep dive into a new definition or theorem. Then come back when you're ready—we'll be here.

Outline

This book is designed to be read linearly. Part I introduces the category **Poly** and illustrates how it models interaction protocols; while Part II highlights a crucial operation on **Poly**, the composition product, which upgrades the theory so that it properly captures the time evolution of a dynamical system.

Part I (The category of polynomial functors) consists of:

- Chapter 1 (Representable functors from the category of sets), in which we review category-theoretic constructions on sets and the Yoneda lemma;
- Chapter 2 (Polynomial functors), in which we introduce our objects of study and present several perspectives from which to view them;
- Chapter 3 (The category of polynomial functors), in which we define **Poly** by introducing its morphisms and demonstrate many ways to work with them, including how they model interaction protocols;
- Chapter 4 (Dynamical systems as dependent lenses), in which we use a specific class of morphisms in **Poly** to model discrete-time dynamical systems; and
- Chapter 5 (More categorical properties of polynomials), in which we describe a smorgasbord of additional category-theoretic structures on **Poly**.

Part II (A different category of categories) consists of:

- Chapter 6 (The composition product), in which we examine a monoidal structure on **Poly** given by substituting one polynomial into another;
- Chapter 7 (Polynomial comonoids and retrofunctors), in which we show that the category of comonoids in **Poly** with respect to the composition product is equivalent to a category of small categories we call $\mathbf{Cat}^\sharp$ whose morphisms are not functors;
- Chapter 8 (Categorical properties of polynomial comonoids), in which we study the structure and utility of $\mathbf{Cat}^\sharp$; and
- Chapter 9 (New horizons), in which we list open questions.

Choices and conventions

Throughout this book, we have chosen to focus on polynomial functors of a single variable on the category of sets. The motivation for this seemingly narrow scope is twofold: to keep matters as concrete and intuitive as possible, with easy access to elements that we can work with directly; and to demonstrate the immense versatility of even this small corner of the theory. Furthermore, the subject of multi-variate polynomials arises by considering what are called comonoids and comodules in **Poly** [Spi23].

Below is a list of conventions we adopt; while it is not comprehensive, any unusual choices are justified within the text, often as a footnote.

The natural numbers include 0, so $\mathbb{N} := \{0, 1, 2, \ldots\}$. Throughout this book, when referring to finite sets, we will adopt the following convention: $\mathsf{0} := \{\} = \varnothing$, $\mathsf{1} := \{1\}$, $\mathsf{2} := \{1, 2\}$, $\mathsf{3} := \{1, 2, 3\}$, and so on, with $\mathsf{n} := \{1, \ldots, n\}$, an n-element set, for each natural number n. For example, in standard font, 5 represents the usual natural number, while in sans serif font, $\mathsf{5}$ represents the 5-element set $\{1, 2, 3, 4, 5\}$. When the same variable name appears in both italicized and sans serif fonts, the italicized variable denotes a natural number and the sans serif variable denotes the corresponding set; for example, if we state that $m \in \mathbb{N}$, then we also understand m to mean the set $\{1, \ldots, m\}$.

The names of categories will be capitalized. We will mostly ignore size issues, but roughly speaking small categories will be written in script (e.g. $\mathcal{C}, \mathcal{D}$), while large categories (usually, but not always, named) will be written in bold (e.g. $\mathbf{Poly}, \mathbf{C}$). We use $\mathbf{Set}$ to denote the category of (small) sets and functions and $\mathbf{Cat}$ to denote the category of (small) categories and functors. We use exponential notation $\mathcal{D}^{\mathcal{C}}$ to denote the category of functors $\mathcal{C} \to \mathcal{D}$ and natural transformations.

We write either $c \in \operatorname{Ob} \mathcal{C}$ or $c \in \mathcal{C}$ to denote an object c of a category $\mathcal{C}$. We use $\sum$ rather than $\coprod$ to denote coproducts. We denote the collection of morphisms $f : c \to d$ in a category $\mathcal{C}$ by using the name of the category itself, followed by the ordered pair of objects: $\mathcal{C}(c, d)$. We denote the domain of f by $\operatorname{dom} f$ and the codomain of f by $\operatorname{cod} f$. We use $:=$ for definitions and temporary assignments, as opposed to $=$ for identifications that can be observed and proven. We use $\cong$ to indicate an isomorphism of objects and $=$ to indicate an equality of objects, although the choice of the former does not preclude the possibility of the latter, nor does the latter necessarily imply any significance beyond an arbitrary selection that has been made. We will freely use the definite article "the" to refer to objects that are unique only up to unique isomorphism.

We list nullary operations before binary ones: for example, we denote a monoidal category $\mathcal{C}$ with monoidal unit I and monoidal product $\odot$ by $(\mathcal{C}, I, \odot)$, or say that $(I, \odot)$ is a monoidal structure on $\mathcal{C}$.

Past, present, and future

The idea for this book began in 2020, originally as part of a joint work with David Jaz Myers on using categories to model dynamical systems. It soon became clear, however, that our writing and his—while intimately related—would be better off as separate volumes. His book is nonetheless an excellent companion to ours: see [Mye22].

In the summer of 2021, we taught a course on a draft of this book that was livestreamed from the Topos Institute. Lecture recordings are freely available at `https://topos.site/poly-course/`. A follow-up workshop with additional recorded lectures and write-ups was held in early 2024; see `https://topos.site/events/poly-at-work/`.

The theory and application of polynomial functors comprise an active area of research. We have laid the foundations here, but work is still ongoing. Even while writing this, we discovered new results and uses for polynomial functors, which only goes to show how bountiful **Poly** can be in its rewards—but of course, we had to cut things off somewhere. We say this in the hope that you will keep the following in mind: where this book ends, the story will have just begun.

Acknowledgments

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

Thanks go to John Baez, Eric Bond, Spencer Breiner, Kris Brown, Matteo Capucci, Valeria de Paiva, Joseph Dorta, Brendan Fong, Richard Garner, Bruno Gavranović, Neil Ghani, Ben Goertzel, Tim Hosgood, Samantha Jarvis, Max Lieblich, Shaowei Lin, Owen Lynch, Joachim Kock, Jérémie Koenig, Sophie Libkind, Joshua Meyers, Dominic Orchard, Nathaniel Osgood, Evan Patterson, Brandon Shapiro, Juliet Szatko, Tish Tanski, Todd Trimble, Adam Vandervorst, Jonathan Weinberger, and Christian Williams.

This material is based upon work supported by the AFOSR under award numbers FA9550-20-1-0348 and FA9550-23-1-0376.

Contents

Part I

The category of polynomial functors

Chapter 1

Representable functors from the category of sets

In this chapter, we lay the categorical groundwork needed to define our category of interest, the category of polynomial functors. We begin by examining a special kind of polynomial functor that you may already be familiar with—representable functors from the category **Set** of sets and functions. We highlight the role these representable functors play in what is arguably the fundamental theorem of category theory, the Yoneda lemma. We will also discuss sums and products of sets and of set-valued functors, which we will need to construct our polynomial functors.

1.1 Representable functors and the Yoneda lemma

Representable functors—special functors to the category of sets—provide the foundation for the category **Poly**. While much of the following theory applies to representable functors from any category, we need only focus on representable functors $\mathbf{Set} \to \mathbf{Set}$.

Definition 1.1 (Representable functor)**.** For a set S, we denote by $y^S \colon \mathbf{Set} \to \mathbf{Set}$ the functor that sends each set X to the set $X^S = \mathbf{Set}(S, X)$ and each function $h \colon X \to Y$ to the function $h^S \colon X^S \to Y^S$ that sends $g \colon S \to X$ to $g \mathbin{⨟} h \colon S \to Y$.[a]

We call a functor (isomorphic to one) of this form a *representable functor*, or a *representable*. In particular, we call y^S the functor *represented by* S, and we call S the *representing set of* y^S. As y^S denotes raising a variable to the power of S, we will also refer to representables as *pure powers*.

[a]Throughout this text, given morphisms $f \colon A \to B$ and $g \colon B \to C$ in a category, we will denote their composite morphism $A \to C$ interchangeably as $f \mathbin{⨟} g$ or $g \circ f$ (or even gf), whichever is more convenient.

The symbol y stands for Yoneda, for reasons we will explain in Lemma 1.10 and Exercise 1.12 #5.

Throughout this book, we will use the notation $\mathsf{0} := \{\} = \varnothing$, $\mathsf{1} := \{1\}$, $\mathsf{2} := \{1, 2\}$, $\mathsf{3} := \{1, 2, 3\}$, and so on, with $\mathsf{n} := \{1, \ldots, n\}$

Example 1.2. The functor that sends each set X to $X \times X$ and each function $h \colon X \to Y$ to $(h \times h) \colon (X \times X) \to (Y \times Y)$ is representable. After all, $X \times X \cong X^{\mathsf{2}}$, so this functor is the pure power y^{2}.

Exercise 1.3 (Solution here). For each of the following functors $\mathbf{Set} \to \mathbf{Set}$, say if it is representable or not; if it is, give the set that represents it.

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

Now that we have introduced representable functors, we study the maps between them. As representables are functors, the maps between them are natural transformations.

Proposition 1.4. 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 its X-component $X^f \colon X^S \to X^R$ is given by sending $g \colon S \to X$ to $f \mathbin{⨾} g \colon R \to X$.

Proof. See Exercise 1.5. □

Exercise 1.5 (Solution here). To prove Proposition 1.4, show that for any function $f \colon R \to S$, the given construction $y^f \colon y^S \to y^R$ really is a natural transformation. That is, for any function $h \colon X \to Y$, show that the following naturality square commutes:

$$\begin{tikzcd} X^S \arrow[r, "h^S"] \arrow[d, "X^f"'] \arrow[dr, phantom, "?"] & Y^S \arrow[d, "Y^f"] \\ X^R \arrow[r, "h^R"'] & Y^R \end{tikzcd} \tag{1.6}$$

◊

Exercise 1.7 (Solution here). Let X be an arbitrary set. For each of the following sets R, S and functions $f: R \to S$, describe the X-component $X^f: X^S \to X^R$ of the natural transformation $y^f: y^S \to y^R$.

1. $R = \mathsf{5}$, $S = \mathsf{5}$, $f = \mathrm{id}_{\mathsf{5}}$. (You should describe the function $X^{\mathrm{id}_{\mathsf{5}}}: X^{\mathsf{5}} \to X^{\mathsf{5}}$.)
2. $R = \mathsf{2}$, $S = \mathsf{1}$, f is the unique function.
3. $R = \mathsf{1}$, $S = \mathsf{2}$, $f(1) = 1$.
4. $R = \mathsf{1}$, $S = \mathsf{2}$, $f(1) = 2$.
5. $R = \mathsf{0}$, $S = \mathsf{5}$, f is the unique function.
6. $R = \mathbb{N}$, $S = \mathbb{N}$, $f(n) = n + 1$. ◊

These representable functors and natural transformations live in the larger category $\mathbf{Set}^{\mathbf{Set}}$, whose objects are functors $\mathbf{Set} \to \mathbf{Set}$ and whose morphisms are the natural transformations between them.

Exercise 1.8 (Solution here). Show that the construction $f \mapsto y^f$ from Proposition 1.4 defines a functor

$$y^-: \mathbf{Set}^{\mathrm{op}} \to \mathbf{Set}^{\mathbf{Set}} \tag{1.9}$$

by verifying functoriality, as follows.

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

We now have all the ingredients we need to state and prove the Yoneda lemma on the category of sets.

Lemma 1.10 (Yoneda lemma)**.** Given a functor $F: \mathbf{Set} \to \mathbf{Set}$ and a set S, there is an isomorphism

$$F(S) \cong \mathbf{Set}^{\mathbf{Set}}(y^S, F) \tag{1.11}$$

where the right hand side is the set of natural transformations $y^S \to F$. Moreover, (1.11) is natural in both S and F.

Proof. Given a natural transformation $m: y^S \to F$, consider its S-component $m_S: S^S \to F(S)$. Applying this function to $\mathrm{id}_S \in S^S$ yields an element $m_S(\mathrm{id}_S) \in F(S)$.

Conversely, given an element $a \in F(S)$, there is a natural transformation we denote by $m^a: y^S \to F$ whose X-component is the function $X^S \to F(X)$ that sends $g: S \to X$ to $F(g)(a)$. In Exercise 1.12 we ask you to show that this is indeed natural in X; that these two constructions, $m \mapsto m_S(\mathrm{id}_S)$ and $a \mapsto m^a$, are mutually inverse; and that the resulting isomorphism is natural. □

Exercise 1.12 (Solution here). In this exercise, we fill in the details of the preceding proof.

1. Show that for any $a \in F(S)$, the maps $X^S \to F(X)$ defined in the proof of Lemma 1.10 are natural in X.
2. Show that the two mappings given in the proof of Lemma 1.10 are mutually inverse, thus defining the isomorphism (1.11).
3. Show that (1.11) as defined is natural in F.
4. Show that (1.11) as defined is natural in S.
5. As a corollary of Lemma 1.10, show that the functor $y^-\colon \mathbf{Set}^{\mathrm{op}} \to \mathbf{Set}^{\mathbf{Set}}$ from (1.9) is fully faithful—in particular, there is an isomorphism $S^T \cong \mathbf{Set}^{\mathbf{Set}}(y^S, y^T)$ for sets S, T. For this reason, we call y^- the *Yoneda embedding*. ◊

How will we go from these representable functors to polynomial ones? Recall that, in algebra, a polynomial is just a sum of pure powers. So we will define a *polynomial functor* **Set** $\to$ **Set** to be a sum of pure power functors—that is, the representable functors y^A for each set A we just introduced.[1]

All of our polynomials will be in one variable, y. Every other letter or number that shows up in our notation for a polynomial will denote a set. For example, in the polynomial

$$\mathbb{R}y^{\mathbb{Z}} + \mathsf{3}y^{\mathsf{3}} + y^A + \sum_{i \in I} y^{R_i + Q_i^2}, \tag{1.13}$$

$\mathbb{R}$ denotes the set of real numbers, $\mathbb{Z}$ denotes the set of integers, $\mathsf{2}$ and $\mathsf{3}$ respectively denote the sets $\{1,2\}$ and $\{1,2,3\}$, and A, I, Q_i, and R_i denote arbitrary sets.

To make sense of these polynomials, we need to define functor addition, both in the binary case (i.e. what is $y^A + y^B$?) and more generally over arbitrary sets (i.e. what is $\sum_{i\in I} y^{A_i}$?). This will allow us to interpret polynomials like (1.13). In particular, just as $3y^3 = y^3 + y^3 + y^3$ in algebra, the summand $\mathsf{3}y^{\mathsf{3}}$ of (1.13) denotes the sum of representables $y^{\mathsf{3}} + y^{\mathsf{3}} + y^{\mathsf{3}}$, while the summand $\mathbb{R}y^{\mathbb{Z}}$ denotes the sum over $\mathbb{R}$ of copies of $y^{\mathbb{Z}}$.

While polynomial functors will be defined as sums, products of polynomials will turn out to be polynomials as well, again mimicking polynomials in algebra. To make sense of these products, we will also define functor multiplication. The construction of sums and products of functors **Set** $\to$ **Set** will rely on the construction of sets and products of sets themselves.

1.2 Sums and products of sets

Let I be a set, and let X_i be a set for each $i \in I$. Classically, we may denote this *I-indexed family of sets* by $(X_i)_{i\in I}$. Categorically, we may view this data as a specific kind of functor: if we identify the set I with the *discrete category* on I, whose objects are the elements of I and whose morphisms are all identities, then $(X_i)_{i\in I}$ can be identified with a functor

[1] This analogy isn't perfect: in algebra, polynomials are generally finite sums of pure powers, whereas our polynomial functors may be infinite sums of representables. However, we are not the first to use the term "polynomial" this way, and the name stuck.

$X: I \to \mathbf{Set}$ with $X(i) := X_i$. To compromise, we will denote an indexed family of sets by $X: I \to \mathbf{Set}$ for a set I viewed as a discrete category (although we will occasionally use the classical notation when convenient), but denote the set obtained by evaluating X at each $i \in I$ by X_i rather than $X(i)$.

To pick out an element of one of the sets in the indexed family $X: I \to \mathbf{Set}$, we need to specify both an index $i \in I$ and an element $x \in X_i$. We call the set of such pairs (i, x) the *sum* of this indexed family, as below.

Definition 1.14 (Sum of sets)**.** Let I be a set and $X: I \to \mathbf{Set}$ be an I-indexed family of sets. The *sum* $\sum_{i \in I} X_i$ *of the indexed family* X is the set

$$\sum_{i \in I} X_i := \{(i, x) \mid i \in I \text{ and } x \in X_i\}.$$

When $I := \{i_1, \ldots, i_n\}$ is finite, we may alternatively denote this sum as

$$X_{i_1} + \cdots + X_{i_n}.$$

Say instead we pick an element from *every* set in the indexed family: that is, we construct an assignment $i \mapsto x_i$, where each $x_i \in X_i$. If every X_i were the same set X, then this would just be a function $I \to X$. More generally, this assignment is what we call a *dependent function*: its codomain X_i *depends* on its input i. We write the signature of such a dependent function as

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

Note that the indexed family of sets $X: I \to \mathbf{Set}$ completely determines this signature. The set of all dependent functions whose signature is determined by a given indexed family of sets is the *product* of that indexed family, as below.

Definition 1.15 (Product of sets)**.** Let I be a set and $X: I \to \mathbf{Set}$ be an I-indexed family of sets. The *product* $\prod_{i \in I} X_i$ *of the indexed family* X is the set of dependent functions

$$\prod_{i \in I} X_i := \{f: (i \in I) \to X_i\}.$$

When $I := \{i_1, \ldots, i_n\}$ is finite, we may alternatively denote this product as

$$X_{i_1} \times \cdots \times X_{i_n} \qquad \text{or} \qquad X_{i_1} \cdots X_{i_n}.$$

For a dependent function $f: (i \in I) \to X_i$, we may denote the element of X_i that f assigns to $i \in I$ by $f(i)$, $f\,i$, or f_i. When $I := \{i_1, \ldots, i_n\}$ is finite, we may identify f with the n-tuple $(f(i_1), \ldots, f(i_n))$; similarly, when $I := \mathbb{N}$, we may identify f with the infinite sequence $(f_0, f_1, f_2, \ldots)$.

Example 1.16. If $I := 2 = \{1, 2\}$, then an I-indexed family $X\colon I \to \mathbf{Set}$ consists of two sets—say $X_1 := \{a, b, c\}$ and $X_2 := \{c, d\}$. Their sum is then the disjoint union

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

The cardinality[a] of $X_1 + X_2$ will always be the sum of the cardinalities of X_1 and X_2, justifying the use of the word "sum."

Meanwhile, their product is the usual cartesian product

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

The cardinality of $X_1 \times X_2$ will always be the product of the cardinalities of X_1 and X_2, justifying the use of the word "product."

[a]The *cardinality* of a set is the number of elements it contains, at least when the set is finite; with care the notion can be extended to infinite sets as well.

Exercise 1.17 (Solution here). Let I be a set.

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

As a special case, suppose that $I := 0 = \varnothing$ and that $X\colon \varnothing \to \mathbf{Set}$ is the unique empty indexed family of sets.

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

The following standard fact describes the constructions from Definitions 1.14 and 1.15 categorically and further justifies why we call them sums and products.

Proposition 1.18. Let I be a set and $X\colon I \to \mathbf{Set}$ be an I-indexed family 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 $X\colon I \to \mathbf{Set}$, viewed as a diagram), 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 $X\colon I \to \mathbf{Set}$, viewed as a diagram).

Proof. The sum $\sum_{i \in I} X_i$ comes equipped with an inclusion $\iota_j\colon X_j \to \sum_{i \in I} X_i$ for each $j \in I$ given by $x \mapsto (j, x)$. The product $\prod_{i \in I} X_i$ comes equipped with a projection $\pi_j\colon \prod_{i \in I} X_i \to X_j$ for each $j \in I$ sending each $f\colon (i \in I) \to X_i$ to $f(j)$. These satisfy the universal properties for categorical coproducts and products, respectively; see Exercise 1.19. □

Exercise 1.19 (Solution here).

1. Show that $\sum_{i \in I} X_i$ along with the inclusions $\iota_j \colon X_j \to \sum_{i \in I} X_i$ described in the proof of Proposition 1.18 satisfy the universal property of a categorical coproduct: for any set Y with functions $g_j \colon X_j \to Y$ for each $j \in I$, there exists a unique function $h \colon \sum_{i \in I} X_i \to Y$ for which $\iota_j \mathbin{⨟} h = g_j$ for all $j \in I$.
2. Show that $\prod_{i \in I} X_i$ along with the projections $\pi_j \colon \prod_{i \in I} X_i \to X_j$ described in the proof of Proposition 1.18 satisfy the universal property of a categorical product: for any set Y with functions $g_j \colon Y \to X_j$ for each $j \in I$, there exists a unique function $h \colon Y \to \prod_{i \in I} X_i$ for which $h \mathbin{⨟} \pi_j = g_j$ for all $j \in I$. ◊

Though we proved above explicitly that **Set** has all small products and coproducts, from here on out, we will assume the standard categorical fact that **Set** is complete (has all small limits) and cocomplete (has all small colimits).

We have constructed categorical sums and products of sets, but we can also construct categorical sums and products of the maps between them: functions.

Definition 1.20 (Categorical sum and product of functions)**.** Let I be a set and $X, Y \colon I \to$ **Set** be I-indexed families of sets. Given a natural transformation $f \colon X \to Y$, i.e. an I-indexed family of functions $(f_i \colon X_i \to Y_i)_{i \in I}$, its *categorical sum* (or *coproduct*) is the function

$$\sum_{i \in I} f_i \colon \sum_{i \in I} X_i \to \sum_{i \in I} Y_i$$

that, given $i \in I$ and $x \in X_i$, sends $(i, x) \mapsto (i, f_i(x))$; while its *categorical product* is the function

$$\prod_{i \in I} f_i \colon \prod_{i \in I} X_i \to \prod_{i \in I} Y_i$$

that sends each $g \colon (i \in I) \to X_i$ to the *composite dependent function* $(i \in I) \to Y_i$, denoted $g \mathbin{⨟} f$ or $f \circ g$, which sends $i \in I$ to $f_i(g(i))$.

When $I \coloneqq \{i_1, \ldots, i_n\}$ is finite, we may alternatively denote this categorical sum and product of functions respectively as[a]

$$f_{i_1} + \cdots + f_{i_n} \qquad \text{and} \qquad f_{i_1} \times \cdots \times f_{i_n}.$$

[a]We will take care to highlight when this notation may clash with a sum (resp. product) of functions with common domain and codomain whose codomain has an additive (resp. multiplicative) structure.

Exercise 1.21 (Solution here).

1. Show that the categorical sum of functions is the one induced by the universal property of the categorical sum of sets. That is, given a set I, two I-indexed families of sets $X, Y \colon I \to$ **Set**, and a natural transformation $f \colon X \to Y$, the function $\sum_{i \in I} f_i \colon \sum_{i \in I} X_i \to \sum_{i \in I} Y_i$ that we called the categorical sum is induced

by the following composite maps for $j \in I$:

$$X_j \xrightarrow{f_j} Y_j \xrightarrow{\iota'_j} \sum_{i \in I} Y_i,$$

where ι'_j is the inclusion. It then follows by a standard categorical argument that the sum is functorial, i.e. that the sum of identities is an identity and that the sum of composites is the composite of sums.

2. Similarly, show that the categorical product of functions is the one induced by the universal property of the categorical product of sets. That is, given the same setup as the previous part, the function $\prod_{i \in I} f_i \colon \prod_{i \in I} X_i \to \prod_{i \in I} Y_i$ that we called the categorical product is induced by the following composite maps for $j \in J$:

$$\prod_{i \in I} X_i \xrightarrow{\pi_j} X_j \xrightarrow{f_j} Y_j.$$

Again, this implies that the product is functorial. ◊

We now highlight some tools and techniques to help us work with sum and product sets.

Exercise 1.22 (Solution here). Let I be a set and $X \colon I \to \mathbf{Set}$ be an indexed family. 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 product of the indexed family 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 to consider what choices must be made to specify an element of such a set. In the following examples, say that we have a set I and an I-indexed family $X \colon I \to \mathbf{Set}$.

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 1.22 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

$$A \coloneqq \sum_{i \in I} \prod_{j \in J(i)} \sum_{k \in K(i,j)} X(i,j,k). \tag{1.23}$$

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 $X(i,j,k)$.

Here the choice of $k \in K(i,j)$ may depend on i and j: different values of i and j may lead to different sets $K(i,j)$.

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

Example 1.24. Let $I \coloneqq \{1,2\}$; let $J(1) \coloneqq \{j\}$ 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 $X(i,j,k) \coloneqq \{x, y\}$ 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 (1.23) specifies a fixed 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 $k \in K(i,j)$ and an element of $X(i,j,k)$.

Exercise 1.25 (Solution here). Consider the set

$$B \coloneqq \prod_{i \in I} \sum_{j \in J(i)} \prod_{k \in K(i,j)} X(i,j,k). \tag{1.26}$$

1. Give the instructions for choosing an element of B as a nested list, like we did for A just below (1.23).
2. With I, J, K, and X as in Example 1.24, how many elements are in B?
3. Write out three of these elements in the style of Example 1.24. ◊

1.3 Expanding products of sums

We will often encounter sums of sets nested within products, as in (1.23) and (1.26). The following proposition helps us work with these; it is sometimes called the *type-theoretic axiom of choice*, but it is perhaps more familiar as a set-theoretic analogue of the *distributive property* of multiplication over addition. While the identity may look foreign, it captures for sets the same process that you would use to multiply multi-digit numbers from grade school arithmetic or polynomials from high school algebra.

Proposition 1.27 (Pushing $\prod$ past $\sum$)**.** For any sets $I, (J(i))_{i \in I}$, and $(X(i,j))_{i \in I, j \in J(i)}$, we have a natural isomorphism

$$\prod_{i \in I} \sum_{j \in J(i)} X(i,j) \cong \sum_{\bar{j} \in \prod_{i \in I} J(i)} \prod_{i \in I} X(i, \bar{j}(i)).^{a} \tag{1.28}$$

[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. First, we construct a map from the left hand set to the 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:[2]

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

This is an element of the right hand set, because $i \mapsto \pi_1 f i$ is a dependent function in $\prod_{i \in I} J(i)$ and $i \mapsto \pi_2 f i$ is a dependent function in $\prod_{i \in I} X(i, \pi_1 f i)$.

Now we go from right to left. An element of the right hand set 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 map this pair to the following element of the left hand set, a dependent function $(i \in I) \to \sum_{j \in J(i)} X(i,j)$:

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

[2] We omit parentheses for function application here and throughout the text for compactness whenever the meaning is clear.

Finally, we verify that the maps are mutually inverse. An element $(\bar{j}, g)$ of the right hand set is sent by one map and then the other to the pair

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

Meanwhile, an element f of the left hand set is sent by one map and then the other to the dependent function

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

As $f i$ is a pair whose components are $\pi_1 f i$ and $\pi_2 f i$, the dependent function above is precisely f. □

When $J(i) = J$ does not depend on $i \in I$, we can simplify the formula in (1.28).

Corollary 1.29. For any sets I, J, and $(X(i,j))_{i \in I, j \in J}$, we have a natural isomorphism

$$\prod_{i \in I} \sum_{j \in J} X(i,j) \cong \sum_{\bar{j}\colon I \to J} \prod_{i \in I} X(i, \bar{j}i), \tag{1.30}$$

where $\bar{j}$ ranges over all (standard, non-dependent) functions $I \to J$.

Proof. Take $J(i) := J$ for all $i \in I$ in (1.28). Then the set $\prod_{i \in I} J(i)$ becomes $\prod_{i \in I} J$ (which we may denote in exponential form by J^I); its elements, dependent functions $\bar{j}\colon (i \in I) \to J(i) = J$, become standard functions $\bar{j}\colon I \to J$. □

It turns out that being able to push $\prod$ past $\sum$ as in (1.28) is not a property that is unique to sets. In general, we refer to a category having this property as follows.

Definition 1.31 (Completely distributive category)**.** A category $\mathbf{C}$ with all small products and coproducts is *completely distributive*[a] if products distribute over coproducts as in (1.28); 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 $\mathbf{C}$, we have a natural isomorphism

$$\prod_{i \in I} \sum_{j \in J(i)} X(i,j) \cong \sum_{\bar{j} \in \prod_{i \in I} J(i)} \prod_{i \in I} X(i, \bar{j}i). \tag{1.32}$$

[a]While our terminology generalizes that of a completely distributive lattice, which has the additional requirement that the category be a poset, it is unfortunately not standard: a completely distributive category refers to a different concept in some categorical literature. We will not use this other concept, so there is no ambiguity.

The term "completely distributive" comes from lattice theory. As such it is consistent with two different extensions to categories that may not be posets. We use it to mean that the category has all sums and products and that products distribute over sums. Other authors use it to mean that the category has all colimits and limits and a sort of distributivity between them.

So Proposition 1.27 states that **Set** is completely distributive. Once we define the category of polynomial functors, we will see that it, too, is completely distributive.

Corollary 1.29 generalizes to all completely distributive categories as well; we state this formally below.

Corollary 1.33. Let **C** be a completely distributive category. For any sets I and J and objects $(X(i,j))_{i \in I, j \in J}$ in **C**, we have a natural isomorphism

$$\prod_{i \in I} \sum_{j \in J} X(i,j) \cong \sum_{\bar{j} \colon I \to J} \prod_{i \in I} X(i, \bar{j}i). \tag{1.34}$$

Proof. Again, take $J(i) := J$ for all $i \in I$ in (1.32). □

Exercise 1.35 (Solution here). Let **C** be a completely distributive category. How is the usual distributive law

$$X \times (Y + Z) \cong X \times Y + X \times Z$$

for $X, Y, Z \in \mathcal{C}$ a special case of (1.32)? ◊

Throughout this book, such as in the exercise below, you will see expressions consisting of alternating products and sums. Using (1.32), you can always rewrite such an expression as a sum of products, in which every $\sum$ appears before every $\prod$.[3] 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 1.36 (Solution here). Let I, $(J(i))_{i \in I}$, and $(K(i,j))_{(i,j) \in IJ}$ 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 $\prod$. ◊

[3]When an expression is written so that every $\sum$ appears before every $\prod$, it is said to be in *disjunctive normal form*.

Now that we understand sums and products of sets, we are ready to explore sums and products of set-valued functors.

1.4 Sums and products of functors Set → Set

Recall that our goal is to define polynomial functors such as $\mathcal{y}^2 + 2\mathcal{y} + 1$ and the maps between them. We have defined representable functors such as $\mathcal{y}^2$, $\mathcal{y}$, and 1; we just need to interpret sums of functors $\mathbf{Set} \to \mathbf{Set}$. But we might as well introduce products of functors at the same time, because they will very much come in handy. Both these concepts generalize to limits and colimits in $\mathbf{Set}^{\mathbf{Set}}$.

Proposition 1.37. The category $\mathbf{Set}^{\mathbf{Set}}$ has all small limits and colimits, and they are computed pointwise. In particular, on objects, 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

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

natural in X.

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

Focusing on the case of coproducts and products, the following corollary is immediate.

Corollary 1.38 (Sums and products of functors $\mathbf{Set} \to \mathbf{Set}$)**.** Given functors $F, G\colon \mathbf{Set} \to \mathbf{Set}$, their categorical coproduct or sum in $\mathbf{Set}^{\mathbf{Set}}$, denoted $F+G$, is the functor $\mathbf{Set} \to \mathbf{Set}$ defined for $X, Y \in \mathbf{Set}$ and $f\colon X \to Y$ by

$$(F+G)(X) \coloneqq F(X) + G(X) \qquad \text{and} \qquad (F+G)(f) \coloneqq F(f) + G(f);$$

while their categorical product in $\mathbf{Set}^{\mathbf{Set}}$, denoted $F \times G$ or FG, is the functor $\mathbf{Set} \to \mathbf{Set}$ defined for $X, Y \in \mathbf{Set}$ and $f\colon X \to Y$ by

$$(F \times G)(X) \coloneqq F(X) \times G(X) \qquad \text{and} \qquad (F \times G)(f) \coloneqq F(f) \times G(f).$$

More generally, given functors $(F_i)_{i \in I}$ indexed over a set I, their categorical coproduct or sum and categorical product in $\mathbf{Set}^{\mathbf{Set}}$, respectively denoted

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

are respectively defined for $X \in \mathbf{Set}$ by

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

and for functions $f\colon X \to Y$ by

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

We also note the special case of initial and terminal objects. Given a set $I \in \mathbf{Set}$, we will also use I to denote the constant functor $\mathbf{Set} \to \mathbf{Set}$ that sends every set to I.

Corollary 1.39 (Initial and terminal functors $\mathbf{Set} \to \mathbf{Set}$)**.** The constant functor $0\colon \mathbf{Set} \to \mathbf{Set}$ is initial in $\mathbf{Set}^{\mathbf{Set}}$, while the constant functor $1\colon \mathbf{Set} \to \mathbf{Set}$ is terminal in $\mathbf{Set}^{\mathbf{Set}}$.

Proof. As the set 0 is initial in $\mathbf{Set}$ (for every set X there is a unique map $0 \to X$), Proposition 1.37 implies that the constant functor 0 is initial in $\mathbf{Set}^{\mathbf{Set}}$. Similarly, as the set 1 is terminal in $\mathbf{Set}$ (for every set X there is a unique map $X \to 1$), Proposition 1.37 implies that the constant functor 1 is terminal in $\mathbf{Set}^{\mathbf{Set}}$. □

Finally, we note that $\mathbf{Set}^{\mathbf{Set}}$ inherits the distributivity of $\mathbf{Set}$.

Proposition 1.40. The category $\mathbf{Set}^{\mathbf{Set}}$ is completely distributive.

Proof. This follows directly from the fact that $\mathbf{Set}$ itself is completely distributive (Proposition 1.27) and the fact that sums and products in $\mathbf{Set}^{\mathbf{Set}}$ are computed pointwise (Corollary 1.38). □

The following exercises justify some notational shortcuts we will use when denoting polynomial functors. First, for any set A and functor $F\colon \mathbf{Set} \to \mathbf{Set}$, we may write an A-indexed sum of copies of F as AF, the product of F and the constant functor A; for instance, $y + y \cong 2y$.

Exercise 1.41 (Solution here). Show that for a set $A \in \mathbf{Set}$ and a functor $F\colon \mathbf{Set} \to \mathbf{Set}$, an A-indexed sum of copies of F is isomorphic to the product of the constant functor A and F:

$$\sum_{a \in A} F \cong AF.$$

(This is analogous to the fact that adding up n copies of number is equal to multiplying that same number by n.) ◊

Similarly, we may wish to write an A-indexed product of copies of F in exponential form as F^A. But since we have already introduced exponential notation for representable functors, this yields two possible interpretations for the functor $\mathbf{Set} \to \mathbf{Set}$ denoted by y^A: as the functor represented by A, or as the A-indexed product of copies of the identity functor $y\colon \mathbf{Set} \to \mathbf{Set}$. In fact, the following exercise shows that there is no ambiguity, as the two interpretations are isomorphic.

Exercise 1.42 (Solution here).

1. Show that for a set $I \in \mathbf{Set}$, an I-indexed product of 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 that multiplying n copies of a number together is equal to raising that same number to the power of n.)
2. Show that the I-indexed product of 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:

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

   (Hint: You may use the fact that following natural isomorphism holds between sets of functions:

   $$\{f\colon I \times A \to X\} \cong \{g\colon I \to X^A\}.$$

   The process of converting a function f in the left hand set to the corresponding function $i \mapsto (a \mapsto f(i, a))$ in the right is known as *currying*.) ◊

Henceforth, given $A \in \mathbf{Set}$ and a functor $F\colon \mathbf{Set} \to \mathbf{Set}$, we define

$$F^A := \prod_{a \in A} F.$$

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 a power of a representable functor can be simplified to a single representable functor.

With these ingredients, we are finally ready to define what a polynomial functor is. We will begin with this definition in the next chapter.

1.5 Summary and further reading

In this chapter, we reviewed the definition of a *representable functor* $y^S\colon \mathbf{Set} \to \mathbf{Set}$ for $S \in \mathbf{Set}$ sending $X \mapsto X^S$. We then stated and proved the *Yoneda lemma*, a foundational result characterizing maps out of these representables: for an arbitrary functor $F\colon \mathbf{Set} \to \mathbf{Set}$, natural transformations $y^S \to F$ are in natural correspondence with elements of $F(S)$.

We then reviewed other categorical constructions in **Set**, many of which carry over to the polynomial functors we introduce in the next chapter. For a set I, we can view it as a discrete category and consider a functor $X\colon I \to \mathbf{Set}$ as an *I-indexed family of sets* comprised of a set X_i for each $i \in I$. An I-indexed family of sets X has a *sum* (or *coproduct*), the set of pairs (i, x) with $i \in I$ and $x \in X_i$; and a *product*, the set of *dependent functions* $f\colon (i \in I) \to X_i$. Such a dependent function sends each $i \in I$ to an element of X_i. These constructions satisfy the universal properties of coproducts and products; moreover, products distribute over coproducts, making **Set** a completely distributive category. All these constructions and properties are inherited by $\mathbf{Set}^{\mathbf{Set}}$, whose limits (including products) and colimits (including coproducts) are computed pointwise: on one object at a time according to limits and colimits in **Set**.

For other introductions to the Yoneda lemma, the category of sets, or both, take your pick of [Pie91; Bor94; Mac98; Lei14; Rie17; FS19; Che22].

1.6 Exercise solutions

Solution to Exercise 1.3.

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

Solution to Exercise 1.5.

To show that (1.6) commutes, we note that by the construction of the components of y^f in the statement of Proposition 1.4, both vertical maps in the diagram compose functions from S with $f\colon R \to S$ on the left; and by Definition 1.1, 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: $(f \mathbin{⨟} g) \mathbin{⨟} h = f \mathbin{⨟} (g \mathbin{⨟} h)$ for all $g\colon S \to X$.

Solution to Exercise 1.7.

In each case, given $f : R \to S$, we can find the X-component $X^f : X^S \to X^R$ of the natural transformation $y^f : y^S \to y^R$ by applying Proposition 1.4, which says that X^f sends each $g\colon S \to X$ to $f \mathbin{⨟} g\colon R \to X$.

1. Here $X^{\mathrm{id}_5}\colon X^5 \to X^5$ is the identity function.
2. If $f : 2 \to 1$ is the unique function, then $X^f : X^1 \to X^2$ 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 $f : 1 \to 2$ sends $1 \mapsto 1$, then $X^f : X^2 \to X^1$ 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 $f : 1 \to 2$ sends $1 \mapsto 2$, then $X^f : X^2 \to X^1$ 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. Here $X^f : X^5 \to X^0 \cong 1$ is the unique function.
6. If $f : \mathbb{N} \to \mathbb{N}$ sends $n \mapsto n + 1$, then $X^f : X^{\mathbb{N}} \to X^{\mathbb{N}}$ sends each $g\colon \mathbb{N} \to X$ to the function $h\colon \mathbb{N} \to X$ defined by $h(n) := g(n + 1)$ for all $n \in \mathbb{N}$. We can think of X^f as removing the first term of an infinite sequence of elements $(g(0), g(1), g(2), \ldots)$ of X to obtain a new sequence $(g(1), g(2), g(3), \ldots)$.

Solution to Exercise 1.8.

1. The fact that $y^{\mathrm{id}_S} : y^S \to y^S$ is the identity is just a generalization of Exercise 1.7 #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 $\mathrm{id}_S \mathbin{⨟} h = h$, so X^{id_S} is the identity natural transformation on X^S. Hence y^{id_S} is the identity on y^S.
2. Fix $f : R \to S$ and $g\colon S \to T$; we wish to show that $y^g \mathbin{⨟} y^f = y^{f ⨟ g}$. It suffices to show componentwise that $X^g \mathbin{⨟} X^f = X^{f⨟g}$ for every set X. Indeed, X^g sends each $h\colon T \to X$ to $g \mathbin{⨟} h$; then X^f sends $g \mathbin{⨟} h$ to $f \mathbin{⨟} g \mathbin{⨟} h = X^{f⨟g}(h)$.

Solution to Exercise 1.12.

1. To check that $X^S \to F(X)$ is natural in X, we verify that the naturality square

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

commutes for all $h\colon X \to Y$. The top map h^S sends any $g\colon S \to X$ to $g \mathbin{⨟} h$ (Definition 1.1), which is then sent to $F(g \mathbin{⨟} h)(a)$ 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 show that the maps $m \mapsto m_S(\mathrm{id}_S)$ and $a \mapsto m^a$ defined in the proof of Lemma 1.10 are mutually inverse. First, we show that for any natural transformation $m\colon y^S \to F$, we have $m^{m_S(\mathrm{id}_S)} = m$. 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, following $\mathrm{id}_S \in S^S$ around this diagram, we have

$$F(g)(m_S(\mathrm{id}_S)) = m_X(g^S(\mathrm{id}_S)) = m_X(\mathrm{id}_S \mathbin{⨟} g) = m_X(g). \tag{1.43}$$

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. It suffices to show that given functors $F, G\colon \mathbf{Set} \to \mathbf{Set}$ and a natural transformation $\alpha\colon F \to G$, the naturality square
$$\begin{tikzcd} \mathbf{Set}^{\mathbf{Set}}(y^S, F) \arrow[r, "\sim"] \arrow[d, "-\,⨾\,\alpha"'] & F(S) \arrow[d, "\alpha_S"] \\ \mathbf{Set}^{\mathbf{Set}}(y^S, G) \arrow[r, "\sim"] & G(S) \end{tikzcd}$$
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 $\alpha_S(m_S(\mathrm{id}_S)) = (m ⨾ \alpha)_S(\mathrm{id}_S)$. This is also where the bottom map sends $m ⨾ \alpha$, so the square commutes.
4. It suffices to show that given a function $g\colon S \to X$, the naturality square
$$\begin{tikzcd} \mathbf{Set}^{\mathbf{Set}}(y^S, F) \arrow[r, "\sim"] \arrow[d, "y^g\,⨾\,-"'] & F(S) \arrow[d, "F(g)"] \\ \mathbf{Set}^{\mathbf{Set}}(y^X, F) \arrow[r, "\sim"] & F(X) \end{tikzcd}$$
commutes. The left map sends any $m\colon y^S \to F$ to $y^g ⨾ m$, which is sent by the bottom map to $(y^g ⨾ m)_X(\mathrm{id}_X) = m_X(X^g(\mathrm{id}_X)) = m_X(g ⨾ \mathrm{id}_X) = m_X(g)$. 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 (1.43).
5. To show that $\mathbf{Set}^{\mathbf{Set}}(y^S, y^T) \cong S^T$, just take $F \coloneqq y^T$ in Lemma 1.10 so that $F(S) \cong S^T$.

Solution to Exercise 1.17.

1. To show that $I \cong \sum_{i\in I} 1$, observe that $x \in 1 = \{1\}$ if and only if $x = 1$, so $\sum_{i \in I} 1 = \{(i, 1) \mid i \in I\}$. Then the 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 show 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 indeed only one such dependent function.
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 $I = 0 = \varnothing$, we have $\sum_{i\in\varnothing} X_i = \sum_{i\in I} 1 \cong I = 0$, where the equation on the left follows from #3 and the isomorphism in the middle follows from #1.
5. As $I = \varnothing$, we have $\prod_{i\in\varnothing} X_i = \prod_{i\in I} 1 \cong 1$, where the equation on the left follows from #3 and the isomorphism on the right follows from #2.

Solution to Exercise 1.19.

1. Any function $h\colon \sum_{i\in I} X_i \to Y$ for which $\iota_j ⨾ h = g_j$ for all $j \in I$ must satisfy $h(j, x) = h(\iota_j(x)) = g_j(x)$ for all $j \in I$ and $x \in X_j$. This uniquely characterizes h, so if we define $h(j, x) \coloneqq g_j(x)$ we are done.
2. Any function $h\colon Y \to \prod_{i\in I} X_i$ for which $h ⨾ \pi_j = g_j$ for all $j \in I$ must satisfy $h(y)_j = \pi_j(h(y)) = g_j(y)$ for all $y \in Y$ and $j \in I$. This uniquely characterizes $h(y)$ and thus h, so if we define $h(y)\colon (i \in I) \to X_i$ to be the dependent function given by $i \mapsto g_i(y)$ we are done.

Solution to Exercise 1.21.

1. It suffices to show that the following square, where the vertical maps are the inclusions, commutes for all $j \in I$:
$$\begin{tikzcd} X_j \arrow[r, "f_j"] \arrow[d, "\iota_j"] & Y_j \arrow[d, "\iota'_j"] \\ \sum_{i\in I} X_i \arrow[r, "\sum_{i\in I} f_i"] & \sum_{i\in I} Y_i \end{tikzcd}$$
Given $x \in X_j$, the left inclusion map sends x to (j, x), which the bottom sum of maps sends to $(j, f_j(x))$. Meanwhile, the top map sends x to $f_j(x)$, which the right inclusion map again sends to $(j, f_j(x))$.

2. It suffices to show that the following square, where the vertical maps are the projections, commutes for all $j \in I$:

$$\begin{tikzcd} \prod_{i\in I} X_i \arrow[r, "\prod_{i\in I} f_i"] \arrow[d, "\pi_j"] & \prod_{i\in I} Y_i \arrow[d, "\pi'_j"] \\ X_j \arrow[r, "f_j"] & Y_j \end{tikzcd}$$

Given $g\colon (i \in I) \to X_i$ in $\prod_{i\in I} X_i$, the top product of maps sends g to $f \circ g$, which the right projection map sends to $f_j(g(j))$. Meanwhile, the left projection map sends g to $g(j)$, which the bottom map again sends to $f_j(g(j))$.

Solution to Exercise 1.22.

1. The second projection $\pi_2(i, x) = x$ sends each pair $p \coloneqq (i, x) \in \sum_{i\in I} X_i$ 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 $S \coloneqq \{s\colon I \to \sum_{i\in I} X_i \mid s \mathbin{⨟} \pi_1 = \mathrm{id}_I\}$ 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 $f(i) \in X_i$ for $i \in I$, so we can define a function $s_f\colon I \to \sum_{i\in I} X_i$ that sends $i \mapsto (i, f(i))$. Then $\pi_1(s_f(i)) = \pi_1(i, f(i)) = i$, so s_f is 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 $i \in I$, so we can write $s(i)$ as an ordered pair $(i, \pi_2(s(i)))$ with $\pi_2(s(i)) \in X_i$. Hence we can define a dependent function $f_s\colon (i \in I) \to X_i$ sending $i \mapsto \pi_2(s(i))$. 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 1.25.

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 $I \coloneqq \{1, 2\}$, $J(1) \coloneqq \{j\}$, $J(2) \coloneqq \{j, j'\}$, $K(1, j) \coloneqq \{k_1, k_2\}$, $K(2, j) \coloneqq \{k_1\}$, $K(2, j') \coloneqq \{k'\}$, and $X(i, j, k) \coloneqq \{x, y\}$ 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 of the summands (resp. factors). 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 1.35.

We wish to show that $X \times (Y+Z) \cong X \times Y + X \times Z$ using (1.32). On the left hand side, we are taking a 2-fold product: a single object times a 2-fold sum. So we should let $I := \mathsf{2}$ and let $J(1) := \mathsf{1}$, with $X(1,1) := X$; and $J(2) := \mathsf{2}$, with $X(2,1) := Y$ and $X(2,2) := Z$. Then

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

where the middle isomorphism follows from (1.32). 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

$$X(1,1) \times X(2,1) + X(1,1) \times X(2,2) \cong X \times Y + X \times Z.$$

Solution to Exercise 1.36.

1. By applying (1.32), 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 (1.32), 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 (1.32), 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 1.41.

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

$$\sum_{a \in A} F(X) \cong (AF)(X)$$

natural in X. The left hand side is the set $\{(a,s) \mid a \in A \text{ and } s \in F(X)\} \cong A \times F(X)$, while the right hand side is also naturally isomorphic to the set $A(X) \times F(X) \cong A \times F(X)$. Alternatively, since $\mathbf{Set}^{\mathbf{Set}}$ is completely distributive by Proposition 1.40, the result also follows from (1.32), with $I := \mathsf{2}, J(1) := A, J(2) := \mathsf{1}, X(1,a) := \mathsf{1}$ (the constant functor) for $a \in A$, and $X(2,1) := F$:

$$AF \cong \left(\sum_{a \in A} \mathsf{1}\right) F \cong \prod_{i \in \mathsf{2}} \sum_{j \in J(i)} X(i,j) \cong \sum_{\bar{j} \in \prod_{i \in \mathsf{2}} J(i)} \prod_{i \in \mathsf{2}} X(i, \bar{j}(i)) \cong \sum_{a \in A} \mathsf{1} \times F \cong \sum_{a \in A} F.$$

Here we used the fact that $A \cong \sum_{a \in A} \mathsf{1}$ from Exercise 1.17 #1 (there we proved the statement for sets, but the same statement for the corresponding constant set-valued functors follows immediately).

Solution to Exercise 1.42.

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 $y^A(X) \cong X^A$, so $\prod_{i \in I} y^A(X) \cong (X^A)^I$, and that $y^{IA}(X) \cong X^{IA}$. By currying, both sides are naturally isomorphic to the set of functions $I \times A \to X$.

Chapter 2

Polynomial functors

In this chapter, we will formally introduce our objects of study: polynomial functors. In addition to the set-theoretic perspective, we will present several more concrete ways to think about polynomials to aid intuition that we will use throughout the rest of this book. We keep the mathematical content of this chapter fairly light, preferring to solidify our conceptual understanding of polynomials, before advancing to deeper categorical content.

2.1 Introducing polynomial functors

Definition 2.1 (Polynomial functor)**.** A *polynomial functor* (or simply *polynomial*) is a functor $p\colon \mathbf{Set} \to \mathbf{Set}$ such that there exists a set I, an I-indexed family of sets $(p[i])_{i\in I}$, and an isomorphism

$$p \cong \sum_{i\in I} y^{p[i]}$$

to the corresponding I-indexed sum of representables.

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

Remark 2.2. Given sets $I, A \in \mathbf{Set}$, it follows from Exercise 1.41 that we have an isomorphism of polynomials

$$\sum_{i\in I} y^A \cong Iy^A.$$

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: $I \cong \sum_{i\in I} 1$.

Exercise 2.3 (Solution here). Consider the polynomial $q := y^8 + 4y$.

1. Does the polynomial q have a representable summand y^2?

2. Does the polynomial q have a representable summand y?
3. Does the polynomial q have a representable summand $4y$? ◊

Example 2.4. Consider the polynomial $p := y^2 + 2y + 1$. It denotes a functor **Set** $\to$ **Set**; where does this functor send the set $X := \{a, b\}$? To be precise, we will rather verbosely say that $I := 4$ and

$$p[1] := 2, \quad p[2] := 1, \quad p[3] := 1, \qquad \text{and} \qquad p[4] := 0$$

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

$$\begin{aligned} p(X) &\cong \sum_{i \in 4} \{a, b\}^{p[i]} \\ &= \{a, b\}^2 + \{a, b\}^1 + \{a, b\}^1 + \{a, b\}^0 \\ &\cong \{(1, (a, a)), (1, (a, b)), (1, (b, a)), (1, (b, b)), (2, (a)), (2, (b)), (3, (a)), (3, (b)), (4, ())\}. \end{aligned}$$

Above, we denote each function $f : p[i] \to \{a, b\}$ in the set $\{a, b\}^{p[i]}$ by the n-tuple $(f(1), \ldots, f(n))$ whenever $p[i] := \mathsf{n}$. For ease of reading, we may drop the parentheses around these n-tuples to obtain the equivalent set

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

As we might expect, the set $p(X)$ contains $2^2 + 2 + 2 + 1 = 9$ elements, equal to the value obtained when we plug $|X| = 2$ into the original polynomial p when we interpret its coefficients and exponents as numbers instead of sets.

In general, a polynomial $p := \sum_{i \in I} y^{p[i]}$ applied to a set X expands to

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

and can be thought of as the set of all pairs comprised of an element of I and a function $p[i] \to X$ or, equivalently, a $p[i]$-tuple of elements of X.

Exercise 2.5 (Solution here). In the verbose style of Example 2.4, write out all the elements of $p(X)$ for p and X as follows (if there are infinitely many, denote the set $p(X)$ some other way):

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

The following proposition shows how the polynomial functor p itself determines the set I over which we sum up representables to obtain p.

Proposition 2.6. 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

$$p \cong \sum_{i \in p(1)} y^{p[i]}. \tag{2.7}$$

Proof. We need to show that $I \cong p(1)$; the latter claim follows directly. In Exercise 1.17 #1, we showed that $I \cong \sum_{i \in I} 1$, so it suffices to show that $(y^{p[i]})(1) \cong 1$ for all $i \in I$. Indeed, $1^{p[i]} \cong 1$ because there is a unique function $p[i] \to 1$ for each $p[i]$. □

We can draw an analogy between Proposition 2.6 and evaluating $p(1)$ for a polynomial p from high school algebra, which yields the sum of the coefficients of p. The notation in (2.7) will be how we denote arbitrary polynomials from now on, and we will use the following terms to denote the sets $p(1)$ and $p[i]$ for $i \in p(1)$ on which a polynomial p depends.

Definition 2.8 (Position and direction)**.** Given a polynomial functor

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

we call an element $i \in p(1)$ a *position of* p or a *p-position*, and we call an element $a \in p[i]$ a *direction of p at i* or a *$p[i]$-direction*. We call $p(1)$ the *position-set of p* and $p[i]$ the *direction-set of p at i*.

Note that the position-set $p(1)$ along with the $p(1)$-indexed family of direction-sets $p[-]\colon p(1) \to \mathbf{Set}$ uniquely characterize a polynomial p up to isomorphism. Throughout this book, we will often specify a polynomial by giving its positions and its directions at each position.

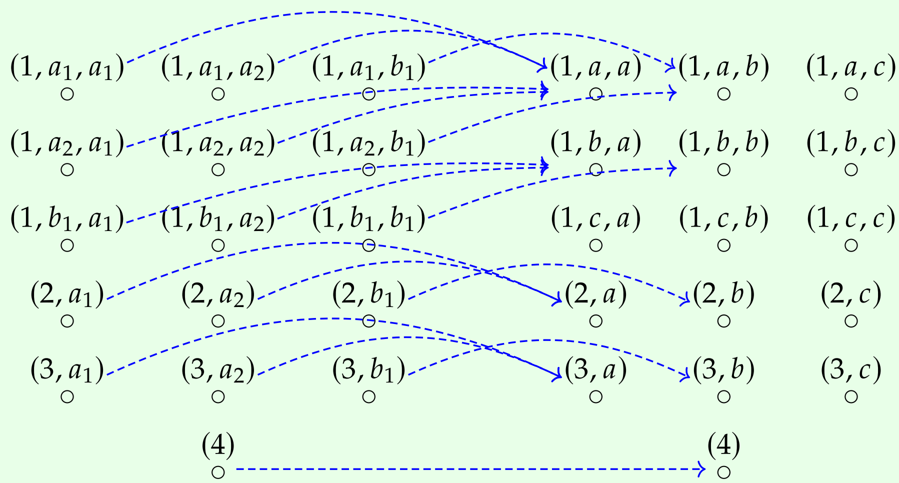
Exercise 2.9 (Solution here). We saw in Proposition 2.6 how to interpret the position-set $p(1)$ of a polynomial p, e.g. $p \coloneqq y^3 + 3y^2 + 4$, as the sum of the coefficients of p: here $p(1) \cong 1 + 3 + 4 \cong 8$. How might you interpret $p(0)$? ◊

As a functor $\mathbf{Set} \to \mathbf{Set}$, a polynomial should act on functions as well as on sets. Below, we explain how.

Proposition 2.10. Let p be an arbitrary polynomial functor, which our notation lets us write as $p \cong \sum_{i \in p(1)} y^{p[i]}$, and let $f\colon X \to Y$ be an arbitrary function. Then $p(f)\colon p(X) \to p(Y)$ sends each $(i, g) \in p(X)$, with $i \in p(1)$ and $g\colon p[i] \to X$, to $(i, g \mathbin{⨾} f)$ in $p(Y)$.

Proof. For each $i \in p(1)$, by Definition 1.1, the functor $y^{p[i]}$ sends f to the function $X^{p[i]} \to Y^{p[i]}$ mapping each $g\colon p[i] \to X$ to $g \fatsemi f\colon p[i] \to Y$. So the sum of these functors over $i \in p(1)$ sends each $(i, g) \in p(X)$ to $(i, g \fatsemi f) \in p(Y)$. $\square$

Example 2.11. 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 2.10, is shown below:



Exercise 2.12 (Solution here). Let $p \coloneqq y^2 + y$. Choose a function $f\colon 1 \to 2$ and write out the induced function $p(f)\colon p(1) \to p(2)$. $\lozenge$

2.2 Special classes of polynomial functors

Here we describe several special classes of polynomials. We have already defined two such special classes: *representables* and *constants*. A *representable polynomial* (or simply a *representable*) is a representable functor, i.e. a polynomial functor isomorphic to y^A for some set A. Meanwhile, a *constant polynomial* (or simply a *constant*) is a constant functor, i.e. a polynomial functor isomorphic to I, interpreted as a functor, for some set I.

Exercise 2.13 (Solution here).

1. Characterize when a polynomial p is *representable* in terms of its positions and/or its directions.
2. Characterize when a polynomial p is *constant* in terms of its positions and/or its directions. $\lozenge$

Like constants, the other two special classes of polynomials we define here will share their names with their algebraic analogues. Throughout, let $p \cong \sum_{i \in p(1)} y^{p[i]}$ be a polynomial functor.

Definition 2.14 (Linear, affine)**.** We say that p is *linear*[a] if $p \cong Iy$ for some set I.

[a]Unlike linear polynomials from high school algebra (which are really *affine linear functions* rather than necessarily *linear functions*), our linear polynomial functors have no (nonzero) constant terms: they always send 0 to 0. A polynomial is *affine* if it is of the form $Ay + B$, though we will not use this concept much in the book.

Definition 2.15 (Monomial)**.** We say that p is a *monomial* if $p \cong Iy^A$ for sets I and A.

Example 2.16. Every constant polynomial $I \cong Iy^0$ is a monomial, as is every linear polynomial $Iy \cong Iy^1$ and every representable $y^A \cong 1y^A$. On the other hand, there are monomials that are neither constant, linear, nor representable, such as $2y^2$ or $\mathbb{N}y^{\mathbb{R}}$. Moreover, there are polynomials that are not monomials, such as $y^4 + 3$ or $\sum_{n \in \mathbb{N}} y^n$.

There is only one polynomial that is both constant and linear, namely $0 \cong 0y$. Similarly, there is only one polynomial (up to isomorphism) that is both constant and representable, namely $1 \cong y^0$. Finally, there is only one polynomial (up to isomorphism) that is both linear and representable, namely the identity functor $1y \cong y \cong y^1$.

In general, every set S has a corresponding constant polynomial S, linear polynomial Sy, and representable polynomial y^S; and as long as $|S| \geq 2$, these are all distinct.

Exercise 2.17 (Solution here).

1. Characterize when a polynomial p is *linear* in terms of its positions and/or its directions.
2. Characterize when a polynomial p is a *monomial* in terms of its positions and/or its directions. $\Diamond$

Later on in Section 5.1, we will see how all four of these special classes of polynomials arise from various adjunctions.

2.3 Interpreting positions and directions

Let us make an informal digression on how we will think about positions and directions of polynomials in this book. While this section has little mathematical content, the intuition we build here will guide us as we delve into the deeper theory of polynomials and their applications to modeling interaction.

The main idea is that a *position* is some status that may be held, while the *directions* at each position are the options available when holding that status. While these positions and directions may be imagined abstractly, here we give some concrete examples.

Example 2.18 (Directions as menu options). Consider a representable and thus polynomial functor y^A for a set A. It has 1 position and the elements of A as its directions. We may think of A as a menu of options to choose from.

The menu may consist of dinner options available at a wedding; then the corresponding representable functor could be

$$y^{\{\texttt{chicken}, \texttt{beef}, \texttt{vegetarian}\}};$$

or it may be the menu of a text editor, in which case the representable could be

$$y^{\{\texttt{cut}, \texttt{copy}, \texttt{paste}\}}.$$

In both these cases, there are exactly 3 directions, so there is an isomorphism of representable functors

$$y^{\{\texttt{chicken}, \texttt{beef}, \texttt{vegetarian}\}} \cong y^{\{\texttt{cut}, \texttt{copy}, \texttt{paste}\}}.$$

Similarly, we may interpret the representable y^2 as a 2-option menu. Such menus are ubiquitous in life: yes or no, true or false, heads or tails, 0 or 1. A 1-option menu, represented by $y^1 \cong y$, is also familiar as an unavoidable choice, the only option: "sorry, ya just gotta go through it." Having no options, represented by $y^0 \cong 1$, is when you actually don't get through it: an impossible decision, a "dead end."

In contrast, we may interpret the representable $y^{[0,1]}$ as a menu with an infinite range of options: a slider with one end labeled 0 and the other labeled 1, able to take on any value in between.

For consistency, we will favor the term "direction" over "option" when referring to the elements of A for a summand y^A of a polynomial. Nevertheless, when we think of a polynomial's directions, we will often think of them as options to choose from.

Example 2.18 shows how we may interpret the directions of a single representable summand as options in a menu. By having multiple representable summands—one for each position—a polynomial may capture more general scenarios with a range of possible menus.

Example 2.19 (Modeling with a polynomial). Consider a coin jar with a slot that may be open or closed. When the slot is open, the jar may accept a penny, a nickel, a dime, or a quarter—there are 4 options to choose from. When the slot is closed, the jar may not accept any coins at all—there are 0 options. We may model this scenario with the polynomial

$$y^{\{\texttt{penny}, \texttt{nickel}, \texttt{dime}, \texttt{quarter}\}} + y^0 \cong y^4 + 1.$$

This polynomial has 2 positions, corresponding to the two statuses the slot could take: open or closed. To delineate these positions, we could take advantage of the fact that every singleton set is isomorphic to 1 and that $1y^A \cong y^A$ to rewrite the above polynomial as

$$\{\texttt{open}\}y^{\{\texttt{penny},\texttt{nickel},\texttt{dime},\texttt{quarter}\}} + \{\texttt{closed}\}y^0 \cong y^4 + 1.$$

Exercise 2.20 (Solution here). Give another example of a real-world scenario that may be modeled by a polynomial with more than 1 position. ◊

2.4 Corolla forests

We would like to have graphical depictions of our polynomials to make them easy to visualize. These will take the form of special graphs known as *corolla forests*. We build up to defining them as follows.

Our first definition will be familiar to students of graph theory, although we will add some technical details suited to our purposes.

Definition 2.21 (Rooted tree)**.** A *rooted tree* is a directed acyclic graph with a distinguished vertex called the *root* such that there exists a unique directed path from the root to each vertex.

We allow infinitely and even uncountably many vertices and infinitely and even uncountably many edges incident to each vertex; on the other hand, each pair of vertices is connected by a (necessarily unique) path of finitely many adjacent edges.

Since all our trees will be rooted, we may refer to them simply as *trees*—roots are implied. We will draw our trees with roots at the bottom and other vertices "growing" upward.

The following terminology will be handy when working with our trees; these terms should be familiar, or at the very least they should match your intuition.

Definition 2.22 (Rooted path; height)**.** A *rooted path* is a (directed) path in a rooted tree from its root to any vertex.

Given a vertex of a rooted tree, its *height* is the length of (i.e. number of edges in) the rooted path to that vertex.

In any rooted tree, the root has height 0, the length of the empty rooted path to the root itself; every neighbor of the root has height 1; every neighbor of a vertex of height 1 either is the root or has height 2; and so forth.

Now we can define a special kind of tree that we will use to depict representable functors.

Definition 2.23 (Corolla)**.** A *corolla* is a rooted tree in which every vertex aside from the root has height 1. We call these vertices the *leaves* of the corolla.

The *corolla associated to a representable functor* y^A for $A \in \mathbf{Set}$ is the corolla whose leaves are in bijection with A.

Example 2.24. Here are the corollas associated to various representables:

$$y^1 \cong y \qquad y^5 \qquad y^{10} \qquad y^{20} \qquad y^{40} \qquad y^{[0,1]}$$

In the example above, the roots are indicated by dots (•), and the leaves are indicated by arrows (↑). Because the direction-set of the representable is in bijection with the leaves of the associated corolla, we can think of each leaf as a direction, so it makes sense to draw the leaves as arrows pointing in different directions. Thinking of directions as menu options like in the previous section, we may view these corollas as mini-decision trees, indicating all the possible options we could select.

Example 2.25. The corolla associated to $y^0 \cong 1$ has *no* leaves: it is the rooted tree consisting of one vertex—the root—and no edges.

$$\overset{\bullet}{y^0 \cong 1}$$

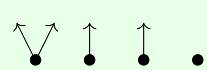
By definition, the root itself is *not* a leaf, so the corolla above does in fact have 0 leaves. With no arrows pointing out, it is the corolla associated to a representable with no directions.

As each representable functor has an associated corolla, each polynomial functor will have an associated disjoint union of corollas that we call a *corolla forest*.

Definition 2.26 (Corolla forest)**.** A *corolla forest* is a disjoint union of corollas.

The *corolla forest associated to a polynomial functor* $p \cong \sum_{i \in p(1)} y^{p[i]}$ is the disjoint union of the corollas associated to each representable summand $y^{p[i]}$ of p. When we draw the corolla forest associated to p, we may say that we are *drawing* p *as a (corolla) forest*. We call the corollas in this forest corresponding to p-positions p*-corollas* and the leaves corresponding to $p[i]$-directions $p[i]$*-leaves*.

Example 2.27. We may draw $p \coloneqq y^2 + 2y + 1$ as a forest like so:



(2.28)

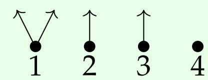
Each of the 4 corollas in (2.28) corresponds to one of the 4 representable summands of

p. The 4 roots in (2.28) correspond to the 4 positions of p, and the leaves connected to each root correspond to the directions at each position. Note that p has 1 position with 2 directions, 2 positions with 1 direction each, and 1 position with 0 directions. Hence (2.28) is the disjoint union of 1 corolla with 2 leaves, 2 corollas with 1 leaf each, and 1 corolla with 0 leaves.

Since $p(1) \cong 4$, we could label the positions of p with the elements of $4 = \{1, 2, 3, 4\}$ so that

$$p[1] = 2, \qquad p[2] = 1, \qquad p[3] = 1, \qquad p[4] = 0.$$

Then we could give these same labels to the roots in (2.28):

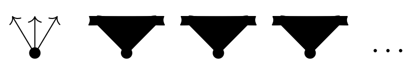


Similarly, we could label the directions and their corresponding leaves, but we will reserve leaf labels for another purpose.

Exercise 2.29 (Solution here). Consider the polynomial $p \coloneqq 2y^3 + 2y + 1$.

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

The position-set or any of the direction-sets of a polynomial may be infinite. This makes their associated corolla forests impossible to draw precisely, but they may be approximated. We sketch the polynomial $y^3 + \mathbb{N}y^{[0,1]}$ as a forest below.



Exercise 2.30 (Solution here). If you were a suitor choosing the corolla forest you love, aesthetically speaking, which would strike your interest? Answer by selecting the associated polynomial:

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

9. Your polynomial's name p here.

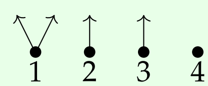
Any reason for your choice? Draw a sketch of your forest. ◊

Corolla forests help us visualize the positions and directions of polynomials, and they will especially come in handy in the next chapter, when we describe the morphisms between our polynomials and how they interact with positions and directions. They may also depict the elements of a polynomial functor applied to a given set, as follows. We have seen that for a polynomial p and a set X,

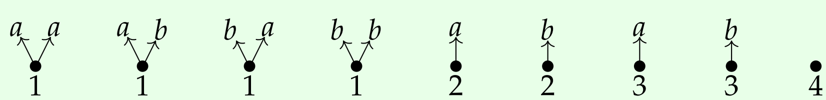
$$p(X) \cong \sum_{i \in p(1)} X^{p[i]} \cong \{(i, f) \mid i \in p(1), f\colon p[i] \to X\}.$$

So an element of $p(X)$ is a p-position i along with a function f that maps each direction at i to an element of X. Equivalently, it is a p-corolla along with a function that maps each of its leaves to an element of X. Then to draw an element $(i, f) \in p(X)$, we simply need to draw the p-corolla corresponding to i and label its leaves with elements of X according to f.

Example 2.31. In Example 2.27, we drew $p := y^2 + 2y + 1$ as a corolla forest like so:



Previously, in Example 2.4, we wrote out all 9 elements of p applied to the set $X := \{a, b\}$ as tuples. We could draw them out instead—an element of $p(X)$ may be depicted as one of the four corollas above with each of its leaves labeled with an element of X:

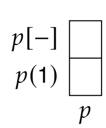


Throughout this book, we will generally use corolla forests to depict polynomials with relatively small numbers of positions or directions, where drawing out entire corolla forests is manageable. Later, we will study how building larger rooted trees out of these corollas corresponds to conducting various categorical operations on our polynomials.

2.5 Polyboxes

Before we conclude this chapter, we introduce one more tool for visualizing polynomials whose full power will not be evident until later.

Throughout this book, we may depict a polynomial p as a pair of boxes stacked on top of each other, like so:



We call this picture the *polyboxes for p*. Think of these boxes as cells in a spreadsheet. The bottom cell, or the *position box*, is restricted to values in the set $p(1)$ (as indicated by the label to its left)—it must be filled with a p-position, say $i \in p(1)$:

$$\begin{array}{r|c|}\cline{2-2} p[-] & \\ \cline{2-2} p(1) & i \\ \cline{2-2} \multicolumn{1}{r}{} & \multicolumn{1}{c}{p} \end{array}$$

The top cell, or the *direction box*, cannot be filled until the position box below it is. Once the position box contains a p-position i, the direction box must be filled with a $p[i]$-direction, say $a \in p[i]$:

$$\begin{array}{r|c|}\cline{2-2} p[-] & a \\ \cline{2-2} p(1) & i \\ \cline{2-2} \multicolumn{1}{r}{} & \multicolumn{1}{c}{p} \end{array}$$

The $p[-]$ label to the left of the direction box reminds us that the a within it is an element of $p[i]$, where i is the entry in the position box. Once we are accustomed to polyboxes, we will often drop these reminder labels, so that

$$\begin{array}{|c|}\hline a \\ \hline i \\ \hline \multicolumn{1}{c}{p} \end{array}$$

serves as a graphical shorthand for the statement "consider a polynomial functor p with position $i \in p(1)$ and direction $a \in p[i]$."

Viewing polynomials as these restricted two-cell spreadsheets reinforces the idea that directions are like menu options: imagine a dropdown menu for the direction box above a filled position box that lists all the directions to choose from at the given position. Polyboxes also help us conceptualize the possible pairs of positions and directions of a polynomial whose corolla forest is impractical to draw, as suggested by the following example.

Example 2.32. Consider the polynomial

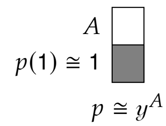
$$p \coloneqq \sum_{r \in \mathbb{R}} y^{[-|r|,|r|]},$$

whose positions are the real numbers and whose directions at position r are the real numbers with magnitude at most $|r|$. There is no clear way to draw p as a corolla forest, but we could draw its polyboxes

$$\begin{array}{r|c|}\cline{2-2} p[-] & s \\ \cline{2-2} p(1) & r \\ \cline{2-2} \multicolumn{1}{r}{} & \multicolumn{1}{c}{p} \end{array}$$

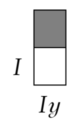
with the condition that r and s are real numbers satisfying $|s| \leq |r|$.

We may also use polyboxes to highlight our special classes of polynomials. When a position box may only be filled with one possible entry, we shade it in like so:

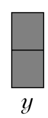


The idea is that if there is only one entry that could fill a given box, then it should come pre-filled—no further choice needs to be made to fill it. Here $p(1) \cong 1$, so p is *representable*; indeed, $p \cong y^A$, where A is the set of possible entries for the unfilled direction box.

Similarly, the polyboxes for a *linear* polynomial Iy, whose direction-set at each position is a singleton, can be drawn like so:

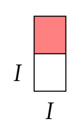


No matter what fills the position box, there is exactly one entry that could fill the direction box, so it comes pre-filled. The identity polynomial functor y, which is both representable and linear, therefore has the following polyboxes:

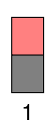


It has exactly one position and exactly one direction, so both its boxes come pre-filled.

Finally, a *constant* polynomial I for some set I has empty direction-sets. We indicate this by coloring its direction box red:



Because every direction-set is empty, there is nothing that may be written in the direction box. The red suggests a kind of error—the direction box cannot be filled. The polynomial functor 1, which is both representable and constant, therefore has the following polyboxes:



In the next chapter, we will introduce the morphisms between polynomial functors and see how their behavior may be depicted using polyboxes.

2.6 Summary and further reading

In this chapter, we introduced the main objects of study in this book: *polynomial functors*, which are sums of representable functors **Set** → **Set**. We write a polynomial p as $\sum_{i \in p(1)} y^{p[i]}$, calling the elements of $p(1)$ the *positions* of p and the elements of $p[i]$ the *directions* of p at position i. As a polynomial is determined up to isomorphism by its position-set and direction-sets, we can think of the data of a polynomial as an indexed family of sets $(p[i])_{i \in p(1)}$.

We highlighted four special classes of polynomials (here I and A are sets):

- constants I, whose direction-sets are all empty;
- linear polynomials Iy, whose direction-sets are all singletons;
- representables y^A, whose position-sets are singletons;
- monomials Iy^A, whose direction-sets all have the same cardinality.

Throughout this book, we will use polynomials to model decision-making agents that hold positions and take directions from those positions. We can draw a polynomial p graphically as a *corolla forest*, with a *corolla* (a rooted tree whose non-root vertices are all leaves) for every p-position i that has a leaf for every $p[i]$-direction. We can also depict a polynomial as a *polybox picture*, resembling two stacked cells in a spreadsheet, to be filled in with an element of i below and an element of $p[i]$ above.

There are many fine sources on polynomial functors. Some of the computer science literature is more relaxed about what a polynomial is. For example, the "coalgebra community" often defines a polynomial to include finite power sets (see e.g. [Jac17]). Other computer science communities use the same definition of polynomial, but refer to it as a *container* and use different words for its positions (they call them "shapes") and directions (they call them, rather unfortunately, "positions"). See e.g. [Abb03; AAG05].

But the notion of polynomial functors seems to have originated from André Joyal. A good introduction to polynomial functors—including an extensive bibliography of references—can be found in [GK12] and more extensive notes in [Koc]; in particular the related work section on page 3 provides a nice survey of the field. A reader may also be interested in the Workshops on Polynomial Functors organized by the Topos Institute: `https://topos.site/p-func-workshop/`.

2.7 Exercise solutions

Solution to Exercise 2.3.

1. No, q does not have y^2 as a representable summand.
2. Yes, q does have y as a representable summand.
3. No, q does not have $4y$ as a representable summand, because $4y$ is not a representable functor! But to make amends, we could say that $4y$ is a *summand*; this means that there is some q' such that $q \cong q' + 4y$, namely $q' \coloneqq y^8$. So $3y$ is also a summand, but y^2 and $5y$ are not.

Solution to Exercise 2.5.

1. Let $I \coloneqq 1$ and $p[1] \coloneqq 3$ so that $p \coloneqq y^3 \cong \sum_{i \in I} y^{p[i]}$. Then

$$p(X) \cong \{(1,4,4,4),(1,4,4,9),(1,4,9,4),(1,4,9,9),(1,9,4,4),(1,9,4,9),(1,9,9,4),(1,9,9,9)\}.$$

2. Let $I := 4$, $p[1] := p[2] := p[3] := 2$, and $p[4] := 1$, so that $p := 3y^2 + 1 \cong \sum_{i \in I} y^{p[i]}$. Then $p(X) \cong \{(1, a, a), (2, a, a), (3, a, a), (4)\}$.
3. Let $I := 0$ so that $p := 0 \cong \sum_{i \in I} y^{p[i]}$. Then $p(X) \cong 0$. Alternatively, note that 0 is the constant functor that sends every set to 0.
4. Let $I := 4$ and $p[i] := 0$ for every $i \in I$ so that $p := 4 \cong \sum_{i \in I} y^{p[i]}$. Then $p(X) \cong \{(1), (2), (3), (4)\} \cong 4$. Alternatively, note that 4 is the constant functor that sends every set to 4.
5. Let $I := 1$ and $p[1] := 1$ so that $p := y \cong \sum_{i \in I} y^{p[i]}$. So $p(X) \cong \{(1, n) \mid n \in \mathbb{N}\} \cong \mathbb{N}$. Alternatively, note that y is the identity functor, so it sends $\mathbb{N}$ to itself.

Solution to Exercise 2.9.

We consider $p(0)$ for arbitrary polynomials p. A representable functor y^S for $S \in \mathbf{Set}$ 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

$$p(0) \cong \sum_{i \in p(1)} \left(y^{p[i]}\right)(0) \cong \sum_{\substack{i \in p(1), \\ p[i] \neq 0}} 0 + \sum_{\substack{i \in p(1), \\ p[i] = 0}} 1 \cong \{i \in p(1) \mid p[i] = 0\}.$$

That is, $p(0)$ is the set of *constant* positions of p, the positions of p that have no directions. For example, if $p := y^3 + 3y^2 + 4$, then $p(0) = 4$. In the language of high school algebra, we might call $p(0)$ the *constant term* of p.

Solution to Exercise 2.12.

We have

$$p(1) \cong \{(1, 1, 1), (2, 1)\} \qquad \text{and} \qquad p(2) \cong \{(1, 1, 1), (1, 1, 2), (1, 2, 1), (1, 2, 2), (2, 1), (2, 2)\}.$$

Say we choose the function $f \colon 1 \to 2$ that sends $1 \mapsto 1$. Then $p(f)$ would send $(1, 1, 1) \mapsto (1, 1, 1)$ and $(2, 1) \mapsto (2, 1)$. If we had instead picked $1 \mapsto 2$ as our function f, then $p(f)$ would send $(1, 1, 1) \mapsto (1, 2, 2)$ and $(2, 1) \mapsto (2, 2)$.

Solution to Exercise 2.13.

1. A polynomial p is representable when $p \cong y^A$ for some set A, and y^A has exactly 1 position. Conversely, if a polynomial p has exactly 1 position, then $p(1) \cong 1$, so we may write p (up to isomorphism) as $p \cong \sum_{1 \in 1} y^{p[1]} \cong y^{p[1]}$, which is representable. So a polynomial p is representable if and only if it has exactly 1 position.
2. A polynomial p is constant when $p \cong I$ for some set I, and I has no directions at any of its positions. Conversely, if every direction-set of a polynomial p is empty, then

$$p \cong \sum_{i \in p(1)} y^{p[i]} \cong \sum_{i \in p(1)} y^0 \cong \sum_{i \in p(1)} 1 \cong p(1),$$

i.e. the set $p(1)$ viewed as a constant functor. So a polynomial p is constant if and only if it has exactly 0 directions at each position.

Solution to Exercise 2.17.

1. A polynomial p is linear when $p \cong Iy$ for some set I, and Iy has exactly 1 direction at each position. (Note that this is even true when $p \cong 0y \cong 0$, for then it is true vacuously.) Conversely, if a polynomial p has exactly 1 direction at each position, then $p[i] \cong 1$ for all $i \in p(1)$, so

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

which is linear. So a polynomial p is linear if and only if it has exactly 1 direction at each position.

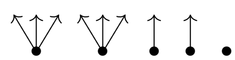
2. A polynomial p is a monomial when $p \cong Iy^A$ for sets I and A, implying that there is an isomorphism of direction-sets $p[i] \cong A \cong p[j]$ for all p-positions i and j (i.e. all the direction-sets of p have the same cardinality). Conversely, if all the direction-sets of a polynomial p are isomorphic to each other, then they are all isomorphic to some set A, so we have
$$p \cong \sum_{i \in p(1)} y^{p[i]} \cong \sum_{i \in p(1)} y^A \cong p(1)y^A,$$
which is a monomial. So a polynomial p is a monomial if and only if all of its direction-sets have the same cardinality.

Solution to Exercise 2.20.

The stopwatch app on my phone has three positions: a `zero` position, from which I may tap a single `start` button; a `running` position, from which I may tap either a `lap` button or a `stop` button; and a `stopped` position, from which I may tap either a `start` button or a `reset` button. Thinking of the buttons available to press as the directions at each position, the corresponding polynomial is
$$\{\texttt{zero}\}y^{\{\texttt{start}\}} + \{\texttt{running}\}y^{\{\texttt{lap}, \texttt{stop}\}} + \{\texttt{stopped}\}y^{\{\texttt{start}, \texttt{reset}\}} \cong y + 2y^2.$$

Solution to Exercise 2.29.

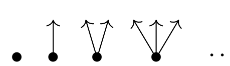
1. Here is the corolla forest associated to $p \coloneqq 2y^3 + 2y + 1$ (note that the order in which the corollas are drawn does not matter):



2. The forest has 5 roots.
3. The roots represent the 5 positions, one position per root.
4. The first and second corollas have 3 leaves each, the third and fourth corollas have 1 leaf each, and the fifth corolla has 0 leaves.
5. The directions at each position correspond to the leaves in each corolla, so just copy the answer from #4, replacing "corolla" with "position" and "leaf" with "direction": the first and second positions have 3 directions each, the third and fourth positions have 1 direction each, and the fifth position has 0 directions.

Solution to Exercise 2.30.

Aesthetically speaking, here is a polynomial that may be drawn as a beautiful corolla forest:
$$p \coloneqq y^0 + y^1 + y^2 + y^3 + \cdots$$
It is reminiscent (and formally related) to the notion of lists: if A is any set, then $p(A) \cong A^0 + A^1 + A^2 + \cdots$ is the set $\mathsf{List}(A)$ of lists (i.e. finite ordered sequences) with entries in A. Here is a picture of the lovely forest associated to p:



Chapter 3

The category of polynomial functors

In this chapter, we will define **Poly**, our main category of interest, so that we have a firm foundation from which to speak about interactive systems. The objects of **Poly** are the polynomial functors that we defined in the previous chapter. Here we will examine the morphisms of **Poly**: natural transformations between polynomial functors. Along the way, we will present some of **Poly**'s most versatile categorical properties.

3.1 Dependent lenses between polynomial functors

Before we define the category **Poly** of polynomial functors, we note that polynomial functors live inside a category already: the category $\mathbf{Set}^{\mathbf{Set}}$ of functors $\mathbf{Set} \to \mathbf{Set}$, whose morphisms are natural transformations. This leads to a natural definition of morphisms between polynomial functors, from which we can derive a category of polynomial functors for free. We call such a morphism a *dependent lens*, or a *lens* for short. If you are familiar with lenses from functional programming, we'll see in Example 3.41 how our notion of a dependent lens is related.

Definition 3.1 (Dependent lens, **Poly**)**.** Given polynomial functors p and q, a *dependent lens* (or simply *lens*) *from* p *to* q is a natural transformation $p \to q$. Then **Poly** is the category whose objects are polynomial functors and whose morphisms are dependent lenses.

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 (isomorphic to) polynomials, but keep all the same morphisms between the objects that remain.

Unraveling the familiar definition of a natural transformation, a dependent lens between polynomial functors $p \to q$ thus consists of a function $p(X) \to q(X)$ for every set X such that naturality squares commute. That is a lot of data to keep track of! Fortunately, there is a much simpler way to think about these lenses, which we will discover using the Yoneda lemma.

Exercise 3.2 (Solution here). Given a set S and a polynomial q, show that a lens $y^S \to q$ can be naturally identified with an element of the set $q(S)$. That is, show that there is an isomorphism

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

natural in both S and q. Hint: Use the Yoneda lemma (Lemma 1.10). ◊

The above exercise gives us an alternative characterization for lenses out of representable functors. But before we can characterize lenses out of polynomial functors in general, we need to describe how coproducts work in **Poly**. Fortunately, since polynomial functors are defined as coproducts of representables, coproducts in **Poly** are easy to understand.

Proposition 3.3. The category **Poly** has all small coproducts, coinciding with coproducts in $\mathbf{Set}^{\mathbf{Set}}$ given by the operation $\sum_{i \in I}$ for each set I.

Proof. By Corollary 1.38, the category $\mathbf{Set}^{\mathbf{Set}}$ has all small 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 an I-indexed family of polynomials $(p_i)_{i \in I}$, its coproduct is

$$\sum_{i \in I} p_i \cong \sum_{i \in I} \sum_{j \in p_i(1)} y^{p_i[j]} \cong \sum_{(i,j) \in \sum_{i \in I} p_i(1)} y^{p_i[j]} \tag{3.4}$$

by Corollary 1.38. This coincides with our notion of polynomial addition from high school algebra: just add all the terms together, combining like terms to simplify. Binary coproducts are given by binary sums of functors, appropriately denoted by $+$, while the initial object of **Poly** is the constant polynomial $\mathsf{0}$.

In particular, (3.4) 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.

Crucially, we have the following corollary.

Corollary 3.5. In the category **Poly**, every polynomial p is the coproduct of its representable summands $(y^{p[i]})_{i \in p(1)}$.

In other words, writing p as the sum $\sum_{i \in p(1)} y^{p[i]}$ is not just a coproduct in $\mathbf{Set}^{\mathbf{Set}}$; it is also a coproduct in **Poly** itself.

We are now ready to give our alternative characterization of dependent lenses. Recall that a polynomial $p \cong \sum_{i \in p(1)} y^{p[i]}$ can be uniquely identified with an indexed family $p[-]\colon p(1) \to \mathbf{Set}$, a functor from the set $p(1)$ viewed as a discrete category.

Proposition 3.6. Given polynomials p and q, there is an isomorphism

$$\mathbf{Poly}(p, q) \cong \prod_{i \in p(1)} \sum_{j \in q(1)} p[i]^{q[j]} \tag{3.7}$$

natural in p and q. In particular, a lens $f : p \to q$ can be identified with a pair $(f_1, f^\sharp)$

$$\begin{tikzcd} p(1) \arrow[rr, "f_1"] \arrow[dr, "{p[-]}"'] & \arrow[d, phantom, "\overset{f^\sharp}{\Leftarrow}"] & q(1) \arrow[dl, "{q[-]}"] \\ & \mathbf{Set} & \end{tikzcd} \tag{3.8}$$

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

Proof. We have $p \cong \sum_{i \in p(1)} y^{p[i]}$. Then by Corollary 3.5 and the universal property of the coproduct, we have a natural isomorphism

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

Applying Exercise 3.2 (i.e. the Yoneda lemma) and the fact that $q \cong \sum_{j \in q(1)} y^{q[j]}$ yields the natural isomorphism

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

so (3.7) follows.

The right hand side of (3.7) is the set of dependent functions $f : (i \in p(1)) \to \sum_{j \in q(1)} p[i]^{q[j]}$. Each such dependent function is uniquely determined by its two projections $\pi_1 f : (i \in p(1)) \to q(1)$ and $\pi_2 f : (i \in p(1)) \to p[i]^{q[\pi_1 f i]}$. These can be identified respectively with a (non-dependent) function $f_1 \coloneqq \pi_1 f$ with signature $p(1) \to q(1)$ and a natural transformation $f^\sharp : q[f_1(-)] \to p[-]$ whose i-component for $i \in p(1)$ is $f_i^\sharp \coloneqq \pi_2 f i \in p[i]^{q[f_1 i]}$. □

We have now greatly simplified our characterization of a dependent lens $f : p \to q$: rather than infinitely many functions satisfying infinitely many naturality conditions, f may simply be specified by a function $f_1 : p(1) \to q(1)$ and, for each $i \in p(1)$, a function $f_i^\sharp : q[f_1 i] \to p[i]$, without any additional restrictions. This characterization can be expressed entirely in the language of positions and directions: f_1 is a function from p-positions to q-positions, while $f_i^\sharp$ for a p-position i is a function from $q[f_1 i]$-directions to $p[i]$-directions. This leads to the following definition.

Definition 3.9 (On-positions function, on-directions map and function)**.** Given a lens $f\colon p \to q$, let $(f_1, f^\sharp)$ denote the pair identified with f via Proposition 3.6. Then we call the function $f_1\colon p(1) \to q(1)$ the *(forward) on-positions function of f*, while we call the natural transformation $f^\sharp\colon q[f_1(-)] \to p[-]$ the *(backward) on-directions map of f*. For $i \in p(1)$, we call the i-component $f_i^\sharp\colon q[f_1 i] \to p[i]$ of $f^\sharp$ the *(backward) on-directions function of f at i*.

The above definition highlights the bidirectional nature of a lens $f\colon p \to q$: it consists of a function going *forward* on positions, following the direction of f from p to q, as well as functions going *backward* on directions, opposing the direction of f from q to p. This forward-backward interaction is what drives the applications of **Poly** we will study.

We prefer to call a morphism between polynomial functors a "lens" rather than a "natural transformation" because we wish to emphasize this concrete on-positions and on-directions perspective. Whenever we do need to view a morphism in **Poly** as a natural transformation, we will refer to them as such.

In the next several sections, we will give some examples of lenses and intuition for thinking about them in terms of interaction protocols, corolla forests, and polyboxes.

3.2 Dependent lenses as interaction protocols

Here is our first example of a dependent lens and a real-world interaction it might model.

Example 3.10 (Modeling an interaction protocol with a lens)*.* Recall our coin jar polynomial from Example 2.19:

$$q := \{\texttt{open}\}y^{\{\texttt{penny},\texttt{nickel},\texttt{dime},\texttt{quarter}\}} + \{\texttt{closed}\}y^{0}.$$

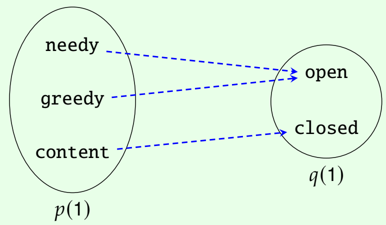
It has 2 positions: its `open` position has 4 directions representing the 4 denominations of coins it may take, while its `closed` position has 0 directions to indicate that it cannot take anything.

Now imagine that we model the owner of this coin jar with the following polynomial:

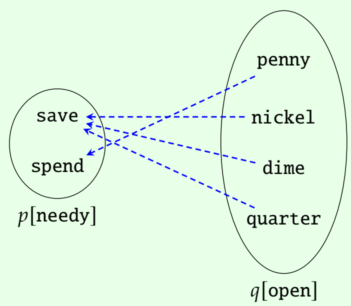
$$\begin{aligned} p := {} & \{\texttt{needy}\}y^{\{\texttt{save},\texttt{spend}\}} \\ & + \{\texttt{greedy}\}y^{\{\texttt{accept},\texttt{reject},\texttt{ask for more}\}} \\ & + \{\texttt{content}\}y^{\{\texttt{count},\texttt{rest}\}}. \end{aligned}$$

Each of its 3 positions represents a possible mood of the owner, and the directions at each position represent the options available to an owner in the corresponding mood. We will construct a lens $f\colon p \to q$ to model the interaction between the owner and their coin jar.

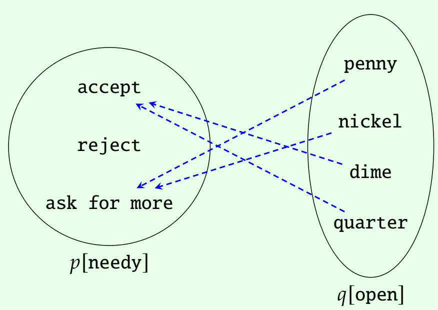
Say that a `needy` or `greedy` owner will keep their coin jar `open`, while a `content` owner will keep their coin jar `closed`. We can express this with an on-positions function f_1 from the set of p-positions (on the left) to the set of q-positions (on the right), as follows (the dashed arrows indicate the function assigments):



From there, say that a `needy` owner whose coin jar receives a `nickel` or higher will choose to `save` it, but one whose coin jar receives a `penny` will choose to `spend` it. Meanwhile, a `greedy` owner whose coin jar receives a `penny` or `nickel` will ask for more, but one whose coin jar receives a `dime` or `quarter` will accept it. We can express this behavior with an on-directions map $f^\sharp\colon q[f_1(-)] \to p[-]$. Its `needy`-component is the on-directions function $f^\sharp_{\texttt{needy}}\colon q[f_1(\texttt{needy})] \to p[\texttt{needy}]$ drawn as follows:



Notice that, by keeping the positions and directions of p on the left and those of q on the right, the on-positions function is drawn from left to right, while the on-directions functions must be drawn right to left. The `greedy`-component of $f^\sharp$ is the on-directions function $f^\sharp_{\texttt{greedy}}\colon q[f_1(\texttt{greedy})] \to p[\texttt{greedy}]$ drawn like so:



Finally, since $q[f_1(\texttt{content})] = q[\texttt{closed}] = \mathtt{0}$, the `content`-component of $f^\sharp$ is the vacuously-defined on-directions function $f^\sharp_{\texttt{content}}\colon \mathtt{0} \to p[\texttt{content}]$. Together, the on-positions function f_1 and the on-directions map $f^\sharp$ defined above completely characterize a lens $f: p \to q$ depicting the interaction between the coin jar and its owner.

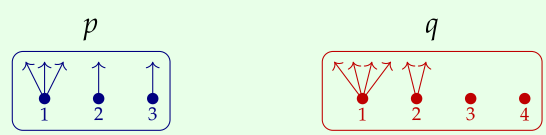
More generally, a lens depicts what we call an *interaction protocol*, a kind of dialogue between two agents regarding their positions and directions. Say that one agent is represented by a polynomial p and another by a polynomial q. Then a lens $f: p \to q$ is an interaction protocol that prescribes how the positions of p influence the positions of q and how the directions of q influence the directions of p. Each p-position $i \in p(1)$ is passed forward via the on-positions function of f to a q-position $f_1 i \in q(1)$. Then each $q[f_1 i]$-direction b is passed backward via the on-directions function of f at i to a $p[i]$-direction $f_i^\sharp b$.

To visualize these lenses, we may use either our corolla forests or our polyboxes.

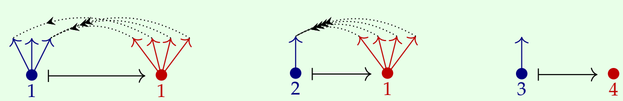
3.3 Corolla forest pictures of dependent lenses

The corolla forest associated to a polynomial concretely depicts its positions and directions, making it easy to extend our corolla forest pictures to depict the dependencies between the positions and directions of two polynomials that a lens between them prescribes.

Example 3.11. Let $p := y^3 + 2y$ and $q := y^4 + y^2 + 2$. We draw them as corolla forests with their positions labeled:



To give a lens $p \to q$, we must send each p-position $i \in p(1)$ to a q-position $j \in q(1)$, then send each direction in $q[j]$ back to one in $p[i]$. We can draw such a lens as follows.



This represents one possible lens $f: p \to q$. The horizontal solid arrows pointing rightward in the picture above tell us that the on-positions function $f_1: p(1) \to q(1)$ is given by

$$f_1(1) := 1, \qquad f_1(2) := 1, \qquad \text{and} \qquad f_1(3) := 4.$$

Then the curved dashed arrows pointing leftward in the picture above describe the on-directions map $f^\sharp: q[f_1(-)] \to p[-]$. On the left, the arrows depict one possible on-directions function $f_1^\sharp: q[1] \to p[1]$ from the 4 directions in $q[1]$ to the 3 directions in $p[1]$. In the middle, the arrows depict the only possible on-directions function

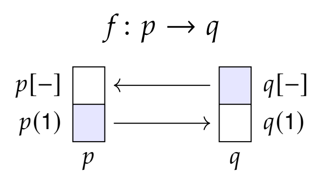
$f_2^\sharp \colon q[1] \to p[2]$ because $|p[2]| = 1$. Finally, on the right, there are no curved arrows, depicting the only possible on-directions function $f_3^\sharp \colon q[4] \to p[3]$ because $|q[4]| = 0$.

Exercise 3.12 (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. Pick an example of a dependent lens $p \to q$ and draw it as we did in Example 3.11.
3. Explain the behavior of your lens as an interaction protocol in terms of positions and directions.
4. Explain in those terms why there can't be any lenses $p \to r$. ◊

3.4 Polybox pictures of dependent lenses

Another way to visualize a dependent lens $f \colon p \to q$ is to draw the polyboxes for p and q.



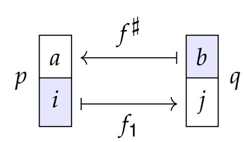
(3.13)

Thinking of the polyboxes as cells in a spreadsheet, the lens prescribes how the values of the cells are computed. The boxes colored blue accept user input, while the other boxes are computed from that input according to the spreadsheet's rules.

The arrows track the flow of information, starting from the lower left. When the user fills the blue position box of p with some $i \in p(1)$, the arrow pointing right indicates that the lens fills the position box of q with some $j \in q(1)$ based on i. The corresponding assignment $i \mapsto j$ is the on-position function f_1 of the lens.

Then when the user fills the blue direction box of q with some $b \in q[j]$, the arrow pointing left indicates that the lens fills the direction box of p with some $a \in p[i]$ based on i and b. Fixing $i \in p(1)$, the assignment $b \mapsto a$ is an on-directions function $f_i^\sharp$ of the lens.

Once all the boxes are filled, we obtain the following:



Here $j := f_1 i$ and $a := f_i^\sharp b$. So a lens is any protocol that will fill the remaining boxes once the user fills the blue boxes, following the directions of the arrows drawn. Be careful: although the arrow $f^\sharp$ is drawn from the codomain's direction box, it also takes into account what is entered into the domain's position box previously. After all, the on-directions map of a lens is dependent on both a position of the domain and a direction of the codomain.

Here is an example of a lens depicted with polyboxes that would be difficult to draw with corolla forests.

Example 3.14 (Modeling with a lens in polyboxes). Caroline asks each of her parents for 20 dollars. Each parent gives Caroline a positive amount of money not exceeding 20 dollars. Caroline spends some of the money she receives before returning the remainder to each parent proportionally according to the amount she received from each.

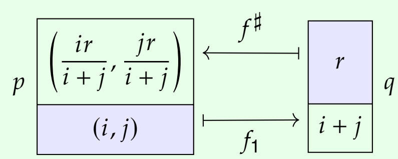
To model this interaction as a lens $f\colon p \to q$, we first define the polynomials p and q. Let

$$p \coloneqq \sum_{(i,j)\in(0,20]\times(0,20]} y^{[0,i]\times[0,j]}$$

be the polynomial that models the parents: its position $(i,j) \in (0,20]\times(0,20]$ consists of the quantities of money that each parent gives to Caroline, and its direction $(i',j') \in [0,i]\times[0,j]$ at that position consists of the quantities of money that Caroline returns to each parent. Then let

$$q \coloneqq \sum_{k\in(0,\infty)} y^{[0,k]}$$

be the polynomial that models Caroline: its position $k \in (0,\infty)$ is the total money that Caroline receives (perhaps Caroline is prepared to receive more money than she is asking for, even if her parents are not prepared to give it), while its direction $r \in [0,k]$ is the money Caroline has remaining after spending some of it. Then we draw the lens f that models their interaction in polyboxes as



We interpret this as saying that the on-positions function f_1 from $p(1) = (0,20]\times(0,20]$ to $q(1) = (0,\infty)$ is defined on $(i,j) \in (0,20]\times(0,20]$ to be

$$f_1(i,j) \coloneqq i+j,$$

while the on-directions function $f^\sharp_{(i,j)}$ from $q[i+j] = [0,i+j]$ to $p[(i,j)] = [0,i]\times[0,j]$ is defined on $r \in [0,i+j]$ to be

$$f^\sharp_{(i,j)}(r) \coloneqq \left(\frac{ir}{i+j}, \frac{jr}{i+j}\right);$$

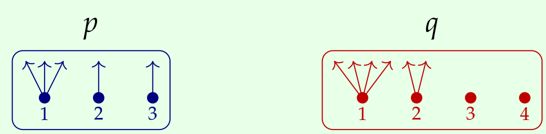
the polybox picture expresses this more compactly. Notice that the position box of q depends only on the position box of p, while the direction box of p depends on the position box of p as well as the direction box of q.

The above example illustrates how we can use polyboxes to specify a particular lens—or, equivalently, how we can use polyboxes to *define* a lens, the same way we might define a function by writing it as a formula in a dummy variable. Later on we will see how polyboxes help depict how lenses compose.

3.5 Computations with dependent lenses

Our concrete characterization of dependent lenses allows us to enumerate them.

Example 3.15 (Enumerating lenses). Let $p := y^3 + 2y$ and $q := y^4 + y^2 + 2$, as in Example 3.11. Again, we draw them as corolla forests with their positions labeled:



How many lenses are there from p to q? The first p-position must be sent to any q-position: 1, 2, 3, or 4. Sending it to 1 would require choosing an on-directions function $q[1] \to p[1]$, or $4 \to 3$; there are 3^4 of these. Similarly, there are 3^2 possible on-directions functions $q[2] \to p[1]$, as well as 3^0 on-directions functions $q[3] \to p[1]$ and 3^0 on-directions functions $q[4] \to p[1]$. Hence there are a total of $3^4 + 3^2 + 3^0 + 3^0 = 92$ ways to choose $f_1(1)$ and $f_1^\sharp$.

The second p-position must also be sent to 1, 2, 3, or 4 before selecting $f_2^\sharp$; there are $1^4 + 1^2 + 1^0 + 1^0 = 4$ ways to do this. Identically, there are 4 ways to choose $f_1(3)$ and $f_3^\sharp$.

In total, there are $92 \cdot 4 \cdot 4 = 1472$ lenses $p \to q$. This coincides with what we obtain by taking the cardinality of both sides of (3.7) and plugging in our values for p and q:

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

Exercise 3.16 (Solution here). For any polynomial p and set A, e.g. $A := 2$, the Yoneda lemma gives an isomorphism $\mathbf{Poly}(y^A, p) \cong p(A)$, so the number of lenses $y^A \to p$ should be equal to the cardinality of $p(A)$.

1. Choose a polynomial p with finitely many positions and directions and draw both y^2 and p as corolla forests.
2. Count all the lenses $y^2 \to p$. How many are there?
3. Compute the cardinality of $p(2)$. Is this the same as the previous answer? $\Diamond$

Exercise 3.17 (Solution here). For each of the following polynomials p, q, compute the number of lenses $p \to q$.

1. $p \coloneqq y^3, \quad q \coloneqq y^4$.
2. $p \coloneqq y^3 + 1, \quad q \coloneqq y^4$.
3. $p \coloneqq y^3 + 1, \quad q \coloneqq y^4 + 1$.
4. $p \coloneqq 4y^3 + 3y^2 + y, \quad q \coloneqq y$.
5. $p \coloneqq 4y^3, \quad q \coloneqq 3y$. $\Diamond$

The following exercises provide alternative formulas for the set of lenses between two polynomials.

Exercise 3.18 (Solution here).

1. Show that the following are isomorphic:
$$\mathbf{Poly}(p, q) \cong \prod_{i \in p(1)} \sum_{j \in q(1)} \prod_{b \in q[j]} \sum_{a \in p[i]} 1. \tag{3.19}$$
2. Show that the following are isomorphic:
$$\mathbf{Poly}(p, q) \cong \sum_{f_1 \colon p(1) \to q(1)} \prod_{j \in q(1)} \mathbf{Set}\left(q[j], \prod_{\substack{i \in p(1), \\ f_1 i = j}} p[i]\right). \tag{3.20}$$
3. Using the language of positions and directions, describe how an element of the right hand side of (3.20) corresponds to a lens $p \to q$. $\Diamond$

In (3.4), we gave an explicit formula for coproducts in **Poly** inherited from coproducts in $\mathbf{Set}^{\mathbf{Set}}$, as justified by Proposition 3.3. We can now use (3.7) to directly verify that the expression on the right hand side of (3.4) satisfies the universal property for the coproduct of polynomials $(p_i)_{i \in I}$ in **Poly**.

Exercise 3.21 (Solution here). Use (3.7) to verify that
$$\mathbf{Poly}\left(\sum_{(i,j) \in \sum_{i \in I} p_i(1)} y^{p_i[j]}, q\right) \cong \prod_{i \in I} \mathbf{Poly}(p_i, q)$$
for all polynomials $(p_i)_{i \in I}$ and q. $\Diamond$

For the remaining exercises in this section, we introduce the concept of the *derivative* $\dot{p}$ of a polynomial p.

Example 3.22 (Derivatives). The *derivative* of a polynomial p, denoted $\dot{p}$, is defined as

$$\dot{p} := \sum_{i \in p(1)} \sum_{a \in p[i]} y^{p[i]-\{a\}}.$$

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

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

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

Since

$$\dot{p}y \cong \sum_{i \in p(1)} \sum_{a \in p[i]} y^{p[i]-\{a\}} y \cong \sum_{i \in p(1)} \sum_{a \in p[i]} y^{p[i]-\{a\}+1} \cong \sum_{i \in p(1)} \sum_{a \in p[i]} y^{p[i]},$$

there exists a canonical lens $\dot{p}y \to p$; you will define this lens in Exercise 3.23. The lens arises in computer science in the context of "plugging in to one-hole contexts"; we will not explore that here, but see [McB01] and [Abb+03] for details.

A lens $f' : p \to \dot{q}$ is similar to a lens $f : p \to q$, except that each p-position explicitly selects a direction of q to remain unassigned. More precisely, while the on-positions function of f sends each p-position to a q-position, the on-positions function of f' sends each p-position i to some $(j, a) \in \sum_{j \in q(1)} q[j]$, picking out not only a q-position j, but also a $q[j]$-direction a. Then the on-directions function of f' at i sends every $q[j]$-direction *other than* a back to a $p[i]$-direction.

Exercise 3.23 (Solution here). The derivative is not very well-behaved categorically, but it is nevertheless intriguing. Take $p, q \in$ **Poly**.

1. Give an explicit construction for the canonical lens $\dot{p}y \to p$ from Example 3.22.
2. Is there always a lens $p \to \dot{p}$? If so, prove it; if not, give a counterexample.
3. Is there always a lens $\dot{p} \to p$? If so, prove it; if not, give a counterexample.
4. Given a lens $p \to q$, is there always a lens $\dot{p} \to \dot{q}$? If so, prove it; if not, give a counterexample.
5. We will define the binary operations $\otimes$ and $[-,-]$ on **Poly** later on in (3.66) and (4.75); and in Exercise 4.80, you will be able to use Exercise 4.78 to deduce that

$$[p, y] \otimes p \cong \sum_{f \in \prod_{i \in p(1)} p[i]} \sum_{i \in p(1)} y^{p(1) \times p[i]}, \tag{3.24}$$

Construct a canonical lens $[p, y] \otimes p \to \dot{p}$.

6. In Example 3.22, we described a lens $p \to \dot{q}$ in terms of "unassigned" directions. Describe a lens $py \to q$ in terms of "unassigned" directions as well. ◊

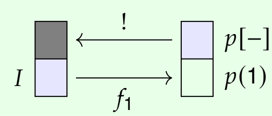
We will not use derivatives very much in the rest of this text, except as shorthand to denote the set of all directions of a polynomial: given a polynomial p, its directions comprise the set $\dot{p}(1)$.

Exercise 3.25 (Solution here). Show that $\dot{p}(1)$ is isomorphic to the set of all directions of p (i.e. the sum of all direction-sets of p). ◊

3.6 Dependent lenses between special polynomials

In Section 2.2, we considered four special classes of polynomials (here I and A are sets): constant polynomials I, linear polynomials Iy, representable polynomials y^A, and monomials Iy^A. Of special note are the constant and linear polynomial 0, the constant and representable polynomial 1, and the linear and representable polynomial y. We now consider lenses with these special polynomials as domains or codomains, highlighting some important examples using polyboxes and leaving most of the rest as exercises. Let p be a polynomial throughout.

Example 3.26 (Lenses from linear polynomials). A lens $f\colon Iy \to p$ can be drawn in polyboxes as follows:



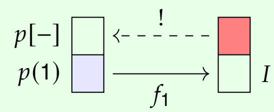
Recall that we shade in the direction box of a linear polynomial to indicate that it can only be filled with one entry. Hence the on-directions map of f is uniquely determined, so f is completely characterized by its on-positions function f_1. We conclude that lenses $Iy \to p$ can be identified with functions $I \to p(1)$.

Exercise 3.27 (Lenses from 0 and y; solution here).

1. Use Example 3.26 to verify that 0 is the initial object of **Poly**.
2. Use Example 3.26 to show that lenses $y \to p$ can be identified with p-positions. ◊

Exercise 3.28 (Lenses to linear polynomials; solution here). Characterize lenses $p \to Iy$ in terms of I and the positions and directions of p. ◊

Example 3.29 (Lenses to constants). A lens $f: p \to I$ can be drawn in polyboxes as follows:



Recall that we color the direction box of a constant red to indicate that it cannot be filled with any entry. Hence the on-directions map of f is again uniquely determined, so f is completely characterized by its on-positions function f_1. (While the on-directions map of f does exist—it is vacuous—it can never produce an element to fill the direction box of p, so we draw it with a dashed line.) We conclude that lenses $p \to I$ can be identified with functions $p(1) \to I$.

Exercise 3.30 (Lenses to 1 and 0; solution here).

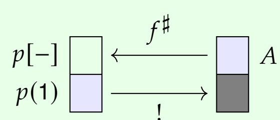
1. Use Example 3.29 to show that 1 is the terminal object of **Poly**.
2. Show that there is exactly one lens whose codomain is 0. What is its domain? ◊

Exercise 3.31 (Lenses from constants such as 1; solution here).

1. Characterize lenses $I \to p$ in terms of I and the positions and directions of p. You may find it helpful to refer to $p(0)$; see Exercise 2.9.
2. Use the previous part to characterize lenses $1 \to p$. ◊

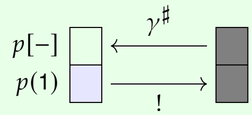
We already know from the Yoneda lemma (see Exercise 3.2) that lenses $y^A \to p$ correspond to elements of $p(A)$, so we understand lenses from representables. Thus we turn our attention to lenses $p \to y^A$.

Example 3.32 (Lenses to representables). A lens $f: p \to y^A$ can be drawn in polyboxes as follows:

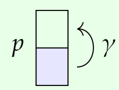


Recall that we shade in the position box of a representable to indicate that it can only be filled with one entry. Hence the on-positions function of f is uniquely determined, so f is completely characterized by its on-directions map $f^\sharp$, which takes a p-position i and a direction $a \in A$ and sends them to a direction $b \in p[i]$. We conclude that lenses $p \to y^A$ can be identified with dependent functions $((i, a) \in p(1) \times A) \to p[i]$.

Example 3.33 (Lenses to y). As a special case of the previous example, a lens $\gamma\colon p \to y$ can be drawn in polyboxes as follows:



Such lenses can be identified with dependent functions $(i \in p(1)) \to p[i]$, which, abusing notation, we also denote by γ. For each p-position i in the blue position box, γ picks out a $p[i]$-direction to fill the unshaded direction box. (Remember that the arrow labeled $\gamma^\sharp$ depends not only on the direction box to its right, but also on the position box of p.) So it makes sense to abbreviate the polybox picture of γ like so:



(3.34)

The correspondence between lenses $p \to y$ and dependent functions $(i \in p(1)) \to p[i]$ exhibited in the previous example also follows directly from (3.7): taking $q \coloneqq y$, we have

$$\mathbf{Poly}(p, y) \cong \prod_{i \in p(1)} \sum_{j \in 1} p[i]^1 \cong \prod_{i \in p(1)} p[i], \tag{3.35}$$

where the right hand side is precisely the set of dependent functions $(i \in p(1)) \to p[i]$. By Exercise 1.22, such functions may be identified with the *sections* of the projection function from $\dot{p}(1) \cong \sum_{i \in p(1)} p[i]$, the set of all directions of p (see Exercise 3.25), to $p(1)$, the set of all positions of p, sending each $(i, a) \in \dot{p}(1)$ to $i \in p(1)$. The fact that this projection determines p (up to isomorphism) motivates the following definition.

Definition 3.36 (Section; bundle)**.** For $p \in \mathbf{Poly}$, a *section* of p is a lens $p \to y$. We denote the set of all sections of p by $\Gamma(p)$; that is,

$$\Gamma(p) \coloneqq \mathbf{Poly}(p, y). \tag{3.37}$$

The *bundle* of p, denoted π_p, is the projection function

$$\dot{p}(1) \cong \sum_{i \in p(1)} p[i] \to p(1)$$

sending $(i, a) \mapsto i$.

With this terminology, we can say that p is determined (up to isomorphism) by its bundle, and that the sections of p can be identified with the sections of its bundle.

To visualize the bundle of p, simply draw it as a corolla forest: a *bundle* of arrows. The bundle projects each leaf down to its root. To visualize a section of p, picture its corollas piled atop each other; a section $\gamma\colon p \to y$ is then a *cross-section* of this pile of p-corollas, picking out an arrow from each one—a direction at each position.

Alternatively, you could think of the arrow curving back to the polyboxes for p in our picture (3.34) of a section $\gamma\colon p \to y$ as *sectioning* off the polyboxes for p from any polyboxes that may otherwise appear to its right. We clarify this intuition by returning to a previous example of a polynomial and considering its sections.

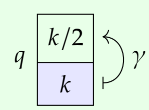
Example 3.38 (Modeling with sections). Recall from Example 3.14 the polynomial

$$q \coloneqq \sum_{k \in (0,\infty)} y^{[0,k]}$$

whose positions $k \in (0, \infty)$ are the possible quantities of money that Caroline receives from her parents and whose directions $r \in [0, k]$ are the possible quantities of money that Caroline has remaining after spending some of it.

One component that was missing from our model was how Caroline spends her money. A section for q fills in this gap by closing the loop from the money Caroline receives to the money she has remaining. Explicitly, a section $\gamma\colon q \to y$ corresponds to a dependent function $(k \in (0, \infty)) \to [0, k]$ that uses the amount of money that Caroline receives to determine the amount of money that she will have remaining.

For instance, if Caroline always spends half the money she receives, then the polyboxes for the section $\gamma\colon q \to y$ that models this behavior can be drawn as follows:



Without γ, we do not know how much money Caroline will decide to spend; having γ makes her decision deterministic and sections this decision off from unknown variables. Of course, the position box of q, the amount of money Caroline receives, is still open to outside influence, as determined by a lens to q such as the one from Example 3.14.

The definition of Γ given in (3.37) makes Γ a functor $\mathbf{Poly} \to \mathbf{Set}^{\mathrm{op}}$ satisfying the following.

Proposition 3.39. The sections functor $\Gamma\colon \mathbf{Poly} \to \mathbf{Set}^{\mathrm{op}}$ sends $(0, +)$ to $(1, \times)$:

$$\Gamma(0) \cong 1 \qquad \text{and} \qquad \Gamma(p + q) \cong \Gamma(p) \times \Gamma(q).$$

Exercise 3.40 (Solution here). Prove Proposition 3.39. ◊

We conclude this section by discussing lenses between monomials, which arise in functional programming.

Example 3.41 (Lenses between monomials are bimorphic lenses). Lenses whose domains and codomains are both monomials are especially simple to write down, because they can be characterized as a pair of (standard, not dependent) functions that are independent of each other, as follows.

Given $I, J, A, B \in \mathbf{Set}$, a lens $f : Iy^A \to Jy^B$ is determined by an on-positions function $f_1 : I \to J$ and an on-directions map: for each $i \in I$, an on-directions function $f_i^\sharp : B \to A$. But the data of such an on-directions map may be repackaged as a single function $f^\sharp : I \times B \to A$. We can do this because every position in I has the same direction-set A, and every position in J has the same direction-set B.

In functional programming, such a pair of functions is called a *bimorphic lens*, or a *lens* for short. In categorical terms, we may say that the monomials in **Poly** span a full subcategory of **Poly** equivalent to *the category of bimorphic lenses*, defined in [Hed18a] (here the category is named after its morphisms rather than its objects). When such a lens arises in functional programming, the two functions that comprise it are given special names:

$$\begin{aligned} \text{get} &:= f_1 : J \to I \\ \text{put} &:= f^\sharp : I \times B \to A \end{aligned} \tag{3.42}$$

Each position $i \in I$ *gets* a position $f_1 i \in J$ and *puts* each direction $b \in B$ back to a direction $f^\sharp(i, b) \in A$.

So a natural transformation between two monomial functors is a bimorphic lens. Then a natural transformation between two polynomial functors is a more general kind of lens: a *dependent* lens, where the direction-sets depend on the positions. Favoring the dependent version, we call these natural transformations *lenses*.

Example 3.43 (Very well-behaved lenses). Consider the monomial Sy^S. Its position-set is S, and its direction-set at each position $s \in S$ is again just S. We could think of each direction as pointing to the 'next' position to move to. We will start to formalize this idea in Example 4.43 and continue this work throughout the following chapters.

Then here is one way we can think of a lens $f : Sy^S \to Ty^T$. Say that Otto takes positions in S, while Tim takes positions in T. Tim will act as Otto's proxy as follows. Tim will model Otto's position via the on-positions function $S \to T$ of f: if Otto is at position $s \in S$, then Tim will be at position $f_1 s \in T$. On the other hand, Otto will take his directions from Tim via the on-directions map $S \times T \to S$ of f: if Tim follows the direction $t' \in T$, then Otto will head from his current position $s \in S$ in the direction $f^\sharp(s, t') \in S$. We interpret these directions as new positions for Otto and Tim to move to. So as Otto moves through the positions in S, he is both modeled and directed by Tim moving through the positions in T.

With this setup, there are three conditions that we might expect the lens $f : Sy^S \to Ty^T$ to satisfy:

1. With Otto at $s \in S$, if Tim stays put at $f_1 s$ (i.e. the direction he selects at $f_1 s$ is still $f_1 s$), then Otto should stay put at s (i.e. the direction he selects at s is still s):

$$f^\sharp(s, f_1 s) = s.$$

2. Once Tim moves to $t \in T$ and Otto moves from $s \in S$ accordingly, Tim's new position should model Otto's new position:

$$f_1(f^\sharp(s, t)) = t.$$

3. If Tim moves to t, then to t', Otto should end up at the same position as where he would have ended up if Tim had moved directly to t' in the first place:

$$f^\sharp(f^\sharp(s, t), t') = f^\sharp(s, t')$$

Such a lens is known to functional programmers as a *very well-behaved lens*; the three conditions above are its *lens laws*. We will see these conditions emerge from more general theory in Example 7.85.

3.7 Translating between natural transformations and lenses

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

- in the language of functors, by specifying a natural transformation $p \to q$, i.e. for each $X \in \mathbf{Set}$, a function $p(X) \to q(X)$ such that naturality squares commute; or
- in the language of positions and directions, 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 how are these two formulations related? Given the data of a lens and that of a natural transformation between polynomials, how can we tell if they correspond to the same morphism? 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 is the crux of the proof of Proposition 3.6, which states that these two formulations of morphisms between polynomials are equivalent; so unraveling these proofs reveals the translation we seek.

Proposition 3.44. Given $p, q \in \mathbf{Poly}$, let $f_1 \colon p(1) \to q(1)$ be a function between their position-sets (like an on-positions function) and $f^\sharp \colon q[f_1(-)] \to p[-]$ be a natural transformation whose components are functions between their direction-sets (like an on-directions map). Then the isomorphism in (3.7) identifies $(f_1, f^\sharp)$ with the natural transformation $f \colon p \to q$ whose X-component $f_X \colon p(X) \to q(X)$ for $X \in \mathbf{Set}$ sends each

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

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

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

Proof. As an element of the product over I on the right hand side of (3.7), the pair $(f_1, f^\sharp)$ is equivalently an I-indexed family of pairs $((f_1 i, f_i^\sharp))_{i \in I}$, where each pair $(f_1 i, f_i^\sharp)$ is an element of

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

By the Yoneda lemma (Lemma 1.10), 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 $f_X^i \colon X^{p[i]} \to q(X)$ given by sending $g \colon p[i] \to X$ to

$$\begin{aligned} q(g)(f_1 i, f_i^\sharp) &= \left(\sum_{j \in q(1)} g^{q[j]} \right)(f_1 i, f_i^\sharp) && \text{(Corollary 1.38)} \\ &= \left(f_1 i, g^{q[f_1 i]}(f_i^\sharp) \right) && \text{(Definition 1.20 and Exercise 1.21)} \\ &= (f_1 i, f_i^\sharp \operatorname{⨾} g). && \text{(Definition 1.1)} \end{aligned}$$

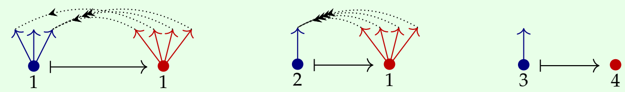
Then the $p(1)$-indexed family of natural transformations $(f^i)_{i \in p(1)}$ is an element of

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

where the isomorphism follows from the universal property of coproducts, as in the proof of Proposition 3.6. Unwinding this isomorphism, we find that $(f^i)_{i \in I}$ corresponds to the natural transformation f from $\sum_{i \in p(1)} y^{p[i]} \cong p$ to q that we desire. □

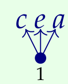
Example 3.45. Let us return once more to the polynomials $p \coloneqq y^3 + 2y$ and $q \coloneqq y^4 + y^2 + 2$

from Example 3.11 and the lens $f\colon p \to q$ depicted below:



Fix a set $X \coloneqq \{a, b, c, d, e\}$. When viewed as a natural transformation, f has as its X-component a function $f_X\colon p(X) \to q(X)$. In other words, for each element of $p(X)$, the lens f should tell us how to obtain an element of $q(X)$.

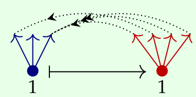
We saw in Example 2.31 that each $(i, g) \in p(X)$ may be drawn as a p-corolla (corresponding to i) whose leaves are labeled with elements of X (according to g). For example, here we draw $(1, g) \in p(X)$, where $g\colon p[1] \to X$ is given by $1 \mapsto c, 2 \mapsto e$, and $3 \mapsto a$:



(3.46)

Similarly, each element of $q(X)$ can be drawn as a q-corolla whose leaves are labeled with elements of X. So what element of $q(X)$ is $f_X(1, g)$?

Proposition 3.44 tells us that $f_X(1, g) = (f_1(1), f_1^\sharp \mathbin{⨟} g)$, so we need only focus on the behavior of f at p-position 1:



To draw $(f_1(1), f_1^\sharp \mathbin{⨟} g)$, we first draw the q-corolla 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 to a $p[1]$-leaf, and use the label there from (3.46) (as prescribed by g). So $f_X(1, g)$ looks like

Proposition 3.44 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: it is the 1-component of f as a natural transformation.

Corollary 3.47. Let p and q be polynomial functors, and let $f\colon p \to q$ be a natural transformation between them. Then the isomorphism in (3.7) sends f to the lens whose on-positions function $f_1\colon p(1) \to q(1)$ is the 1-component of f and whose on-directions map $f^\sharp\colon q[f_1(-)] \to p[-]$ satisfies, for all $i \in p(1)$,

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

Proof. By Proposition 3.44, 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. Moreover, for each $i \in p(1)$, the $p[i]$-component $f_{p[i]}\colon p(p[i]) \to q(p[i])$ of f sends $(i, \mathrm{id}_{p[i]}) \in p(p[i])$ to $(f_1 i, f_i^\sharp \mathbin{⨾} \mathrm{id}_{p[i]}) = (f_1 i, f_i^\sharp)$. □

3.8 Identity lenses and lens composition

Thus far, we have seen how the category **Poly** of polynomial functors and natural transformations can be identified with the category of indexed families of sets and lenses. But in order to actually discuss the latter category, we need to be able to give identity lenses and describe how lenses compose. We can do so by translating between lenses and natural transformations.

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

Exercise 3.48 (Identity lenses; solution here). For $p \in$ **Poly**, 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 $X \in$ **Set** is the identity function on $p(X)$; that is, $(\mathrm{id}_p)_X = \mathrm{id}_{p(X)}$.

Use Corollary 3.47 to show that the on-positions function $(\mathrm{id}_p)_1\colon p(1) \to p(1)$ and the on-directions functions $(\mathrm{id}_p)_i^\sharp\colon p[(\mathrm{id}_p)_1 i] \to p[i]$ for $i \in p(1)$ of id_p are all identity functions. ◊

Similarly, we may infer how two lenses compose by translating them to natural transformations, composing those, then translating back to lenses.

Exercise 3.49 (Composing lenses; solution here). For $p, q, r \in$ **Poly**, 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 $X \in$ **Set** is the composite of the X-components of f and g; that is, $h_X = f_X \mathbin{⨾} g_X$.

Use Proposition 3.44 and Corollary 3.47 to show that the on-positions function $h_1\colon p(1) \to r(1)$ of h is given by $h_1 = f_1 \mathbin{⨾} g_1$, while the on-directions function $h_i^\sharp$ of h for $i \in p(1)$ is given by $h_i^\sharp = g_{f_1 i}^\sharp \mathbin{⨾} f_i^\sharp$. ◊

The following proposition, a restatement of the previous exercise, allows us to interpret commutative diagrams of polynomials in **Poly** in terms of commutative diagrams of their position- and direction-sets in **Set**.

Proposition 3.50. Given $p, q, r \in \mathbf{Poly}$ and lenses $f : p \to q, g : q \to r$, and $h : p \to r$, 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 forward 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 backward on-directions diagram

$$\begin{tikzcd} p[i] & q[f_1 i] \arrow[l, "f_i^\sharp"'] \\ & r[h_1 i] \arrow[ul, "h_i^\sharp"] \arrow[u, "g_{f_1 i}^\sharp"'] \end{tikzcd}$$

commutes in **Set**.

We can use this fact to determine whether a given diagram in **Poly** commutes, as in the following exercise.

Exercise 3.51 (Solution here). Using Proposition 3.50, verify explicitly 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 lenses $\iota : p \to p + q$ and $\kappa : q \to p + q$, then show that for any other polynomial r equipped with lenses $f : p \to r$ and $g : q \to r$, there exists a unique lens $h : p + q \to r$ (shown dashed) making the following diagram commute:

$$\begin{tikzcd} p \arrow[r, "\iota"] \arrow[dr, "f"'] & p + q \arrow[d, dashed, "h"] & q \arrow[l, "\kappa"'] \arrow[dl, "g"] \\ & r & \end{tikzcd} \tag{3.52}$$

◊

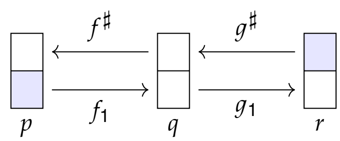
Now that we know how lens composition works in **Poly**, we have a better handle on how it behaves categorically. For instance, we can verify functoriality in **Poly**, as in the following exercise.

Exercise 3.53 (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. Given a topological space X, 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. That is,
$$p_X \coloneqq \sum_{x \in X} y^{\{U \subseteq X \mid x \in U,\, U \text{ open}\}}.$$
Given a continuous map $f \colon X \to Y$, define a lens $p_X \to p_Y$ either by writing down its formula or drawing it in polyboxes.
2. Show that the assignment above defines a functor **Top** → **Poly**.
3. Is this functor full? Is it faithful? ◊

3.9 Polybox pictures of lens composition

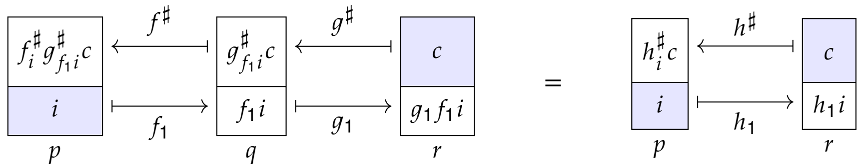
Given lenses $f \colon p \to q$ and $g \colon q \to r$, we can piece their polyboxes together to form polyboxes for their composite, $f \mathbin{⨟} g \colon p \to r$:



The position box for q, which would be blue as part of the polyboxes for $g \colon q \to r$ alone, is instead filled in via f_1; similarly, the direction box for q, which would be blue as part of the polyboxes for just $f \colon p \to q$, is filled in via $g^\sharp$. This forms a spreadsheet-filling protocol that acts as the polyboxes for $f \mathbin{⨟} g$.

As we follow the arrows from left to right and up and left again, take care to note that the arrow $g^\sharp$ depends not only on the direction box of r, but also the position box of q that came before it. Similarly, $f^\sharp$ depends on both the position box of p and the direction box of q. On the other hand, the arrow g_1 depends only on the position box of q, and not the position box of p that came before it: g_1 is the on-positions function for a lens $q \to r$ and therefore depends only on its domain q. (Of course, changing the position box of p may change the position box of q via f_1, thus indirectly affecting what g_1 enters in the position box for r; we mean that if the position box of p changes but the position box of q does not, g_1 will not change the position box of r.) Similarly, $g^\sharp$ does not depend on the position box of p, and $f^\sharp$ does not depend on either box of r. The key is to let each arrow depend on exactly the boxes that come before it in the domain and codomain of the lens that the arrow is a part of.

If we have another lens $h \colon p \to r$, we can interpret the equation $f \mathbin{⨟} g = h$ by filling in their polyboxes and comparing them:

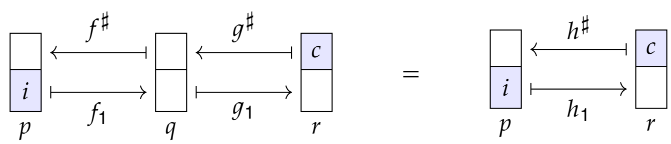


Here we have filled the blue boxes on either side with the same entries. Then if we match up the uncolored boxes in the domain and codomain on either side, we can read off the equations

$$g_1 f_1 i = h_1 i \qquad \text{and} \qquad f_i^\sharp g_{f_1 i}^\sharp c = h_i^\sharp c$$

for every p-position i and $r[h_1 i]$-direction c, which agrees with Exercise 3.49 and Proposition 3.50. Throughout this book, we will often read off equalities of positions and directions from polybox pictures of lens equations in this way.

Note that there is redundancy in the above polybox picture: we have filled in all the boxes for clarity, but their entries are determined by the entries in the blue boxes and the labels on the arrows. So we may omit the entries in the uncolored boxes without losing information, leaving the reader to fill in the blanks:



Remark 3.54. The reader may be concerned that when working with polyboxes, we refer to "spreadsheets" and "protocols" without being rigorous about what they are or what it means to set them equal. We choose to elide this issue to highlight the graphical intuition rather than grinding through the details. This is not to say our work with polyboxes will lack rigor moving forward—if you are particularly worried, you should think of polyboxes as an alternate way to present information about indexed families of sets, dependent functions, and sum and product sets that can be systematically translated—via elementary steps, though perhaps with some laborious bookkeeping—into the more standard $\in$ and $\sum$ and $\prod$ notation we have been using thus far.

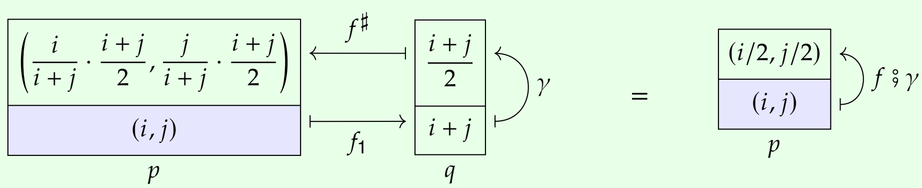
For example, given lenses $f \colon p \to q$ and $g \colon q \to r$, the polyboxes on the left hand side of the equation above should be interpreted as the element of the set

$$\prod_{i \in p(1)} \sum_{k \in r(1)} p[i]^{r[k]} \cong \mathbf{Poly}(p, r)$$

corresponding to the lens $p \to r$ whose on-positions function $p(1) \to r(1)$ is the composite of the on-positions functions f_1 and g_1 and whose on-directions function $r[g_1 f_1 i] \to p[i]$ at $i \in p(1)$ is equal to the composite of the on-directions functions $g_{f_1 i}^\sharp$

and $f_i^\sharp$. In other words, the polyboxes represent the composite lens $f \mathbin{⨟} g$. But the polyboxes show how lenses pass positions and directions back and forth far more legibly than the last two sentences can. Throughout the rest of this book, we will see how this polybox notation provides immediate, reader-friendly computations and justifications; but all these results can be translated back into more grounded mathematical language as desired.

Example 3.55 (Modeling with a composite lens in polyboxes). By composing the lens $f: p \to q$ from Example 3.14 that models the exchange of money between Caroline (modeled by q) and her parents (modeled by p) with the lens $\gamma: q \to y$ from Example 3.38 that models how Caroline spends her money, we obtain a lens $f \mathbin{⨟} \gamma: p \to y$ that models how Caroline's parents spend their money through Caroline. The polybox picture of the composite lens $f \mathbin{⨟} \gamma$ is given by merging the polybox pictures of f and γ:



Here $(i, j) \in p(1) = (0, 20] \times (0, 20]$. The right hand side summarizes what happens to the parents: if the first parent gives away i dollars and the second parent gives away j dollars, eventually the first parent will receive $i/2$ dollars and the second parent will receive $j/2$ dollars. The factored left hand side describes how this happens: the parents give i and j dollars respectively to Caroline, who takes the $i + j$ dollars total and spends half of it. She then returns the remaining half to her parents, splitting the money proportionately according to the amount each parent contributed.

3.10 Symmetric monoidal products of polynomial functors

One of the reasons **Poly** is so versatile is that there is an abundance of monoidal structures on it. Monoidal structures are the key ingredient to many applications of categories to real-world settings, and **Poly** is no different in that regard. As a bonus, if you know how to add and multiply polynomials from high school algebra, then you already know how to compute two of the monoidal products on **Poly**.

We have already seen one of these monoidal structures on **Poly**: the cocartesian monoidal structure, which gives **Poly** its finite coproducts. In fact, we know from Proposition 3.3 that **Poly** has all coproducts: they are given by an operation that looks just like addition. It turns out **Poly** has all products as well, giving it a cartesian monoidal structure that looks just like multiplication.

Proposition 3.56. The category **Poly** has arbitrary products, coinciding with products in $\mathbf{Set}^{\mathbf{Set}}$ given by the operation $\prod_{i \in I}$.

Proof. Unsurprisingly, the proof is very similar to that of Proposition 3.3.

By Corollary 1.38, the category $\mathbf{Set}^{\mathbf{Set}}$ has arbitrary products given by $\prod_{i \in I}$. The full subcategory inclusion $\mathbf{Poly} \to \mathbf{Set}^{\mathbf{Set}}$ reflects these products. It remains to show that **Poly** is closed under the operation $\prod_{i \in I}$.

By Proposition 1.40, $\mathbf{Set}^{\mathbf{Set}}$ is completely distributive. Hence, given polynomials $(p_i)_{i \in I}$, we can use (1.32) to write their product in $\mathbf{Set}^{\mathbf{Set}}$ as

$$\prod_{i \in I} p_i \cong \prod_{i \in I} \sum_{j \in p_i(1)} y^{p_i[j]} \cong \sum_{\bar{j} \in \prod_{i \in I} p_i(1)} \prod_{i \in I} y^{p_i[\bar{j}i]} \cong \sum_{\bar{j} \in \prod_{i \in I} p_i(1)} y^{\sum_{i \in I} p_i[\bar{j}i]}, \tag{3.57}$$

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

Corollary 3.58. 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 $\mathbf{Set}^{\mathbf{Set}}$ (Propositions 3.3 and 3.56) and the fact that $\mathbf{Set}^{\mathbf{Set}}$ itself is completely distributive (Proposition 1.40). □

The result above will allow us to apply (1.32), or sometimes specifically (1.34), to push $\prod$'s past $\sum$'s of polynomials whenever we so desire.

Exercise 3.59 (Solution here). Use (3.7) 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 3.60 (Solution here). Let $p_1 \coloneqq y + 1$, $p_2 \coloneqq y + 2$, and $p_3 \coloneqq y^2$. What is $\prod_{i \in 3} p_i$ according to (3.57)? Is the answer what you would expect? ◊

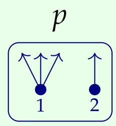
It follows from (3.57) that the terminal object of **Poly** is 1, and that binary products are given by

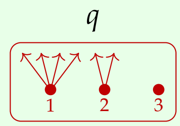
$$p \times q \cong \sum_{i \in p(1)} \sum_{j \in q(1)} y^{p[i] + q[j]}. \tag{3.61}$$

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

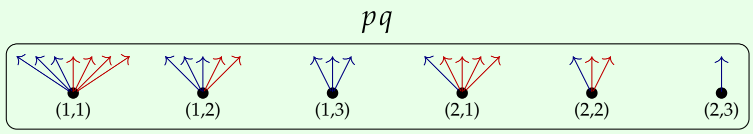
$$pq \coloneqq p \times q.$$

Example 3.62. 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 $pq \cong y^7 + 2y^5 + 2y^3 + y$. 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 3.63 (Solution here).

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

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

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

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

◊

As lenses, the canonical projections $\pi\colon pq \to p$ and $\varphi\colon pq \to q$ behave quite naturally: 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 3.64 (Solution here). Verify that, for $p, q \in \mathbf{Poly}$, the polynomial pq given by (3.61) along with the lenses $\pi\colon pq \to p$ and $\varphi\colon pq \to q$ described above satisfy the universal property of the product of p and q. ◊

Much of Part II will focus on the remarkable features of another monoidal structure, an asymmetric one, whose definition we will postpone—we will save its surprises for when we can better savor them. But here we introduce a third symmetric monoidal structure, given by an operation you were not allowed to do to polynomials back in high school.

Definition 3.65 (Parallel product of polynomials)**.** Let p and q be polynomials. Their *parallel product* (also called *Dirichlet product*), denoted $p \otimes q$, is given by the formula

$$p \otimes q \coloneqq \sum_{i \in p(1)} \sum_{j \in q(1)} y^{p[i] \times q[j]}. \tag{3.66}$$

One should compare this with the formula for the product of polynomials shown in (3.61). The difference is that the parallel product multiplies exponents where the categorical product adds them.

Exercise 3.67 (Solution here).

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

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

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

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

◊

Exercise 3.68 (Solution here).

1. If $p \coloneqq A$ and $q \coloneqq B$ are constant polynomials, what is $p \otimes q$?
2. If $p \coloneqq Ay$ and $q \coloneqq By$ are linear polynomials, what is $p \otimes q$?
3. For arbitrary $p, q \in \mathbf{Poly}$, show that the sets $(p \otimes q)(1)$ and $p(1) \times q(1)$ are isomorphic. ◊

Exercise 3.69 (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. Expand the following expression in the variable y according to the ordinary laws of arithmetic.

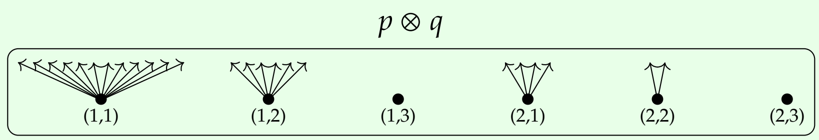
$$(2 \cdot 2^y + 3 \cdot 1^y) \cdot (1 \cdot 4^y + 3 \cdot 3^y)$$

The factors of the above product are called *Dirichlet series*.

4. Describe the connection between the last two parts. (This is why the parallel product $\otimes$ is also known as the *Dirichlet product*.) ◊

Example 3.70. 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 $p \otimes q \cong y^{12} + y^6 + y^4 + y^2 + 2$. We take all pairs of positions, and for each pair we take the product of the directions.



Exercise 3.71 (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. ◊

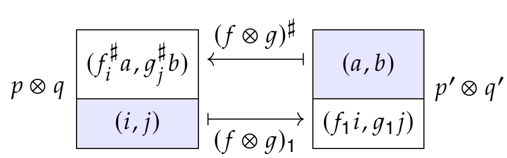
Exercise 3.72 (Solution here). Let $p, q, r \in \mathbf{Poly}$ be any polynomials.

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 3.72, we have gone most of the way to proving that $(\mathbf{Poly}, y, \otimes)$ is a symmetric monoidal category. We sketch the rest of the proof as follows.

Proposition 3.73. The category **Poly** has a symmetric monoidal structure $(y, \otimes)$ where $\otimes$ is the parallel product from Definition 3.65.

Sketch of proof. Given two lenses $f : p \to p'$ and $g : q \to q'$, we need to define a lens $(f \otimes g) : (p \otimes q) \to (p' \otimes q')$. This is easiest to define using polyboxes, keeping in mind that the positions and directions of a parallel product are pairs of positions and directions of its constituent factors:



Here $i \in p(1)$, $j \in q(1)$, $a \in p'[f_1 i]$, and $b \in q'[g_1 j]$.

Then Exercise 3.72 gives us the unitors, associator, and braiding. We have not proven the functoriality of $\otimes$, the naturality of the isomorphisms from Exercise 3.72, 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 3.79. $\square$

Exercise 3.74 (Solution here).

1. What is $(3y^5 + 6y^2) \otimes 4$? Hint: $4 = 4y^0$.
2. Is the class of constant polynomials a $\otimes$-*ideal*; that is, is the parallel product of a polynomial and a constant polynomial always a constant? $\Diamond$

Exercise 3.75 (Solution here). Which of the following special classes of polynomials are closed under $\otimes$? Note also whether they contain y.

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

Exercise 3.76 (Solution here). What is the smallest class of polynomials that is closed under $\otimes$ and contains y? $\Diamond$

Exercise 3.77 (Solution here). Show that for any $p_1, p_2, q \in \mathbf{Poly}$ there is an isomorphism

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

$\Diamond$

Remark 3.78. 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.

There is a more general way to obtain monoidal structures on **Poly** like $\times$ and $\otimes$ using a construction known as the *Day convolution*, defined by a special kind of colimit known as a *coend*. If you have not seen the Day convolution or coends before, do not fret: we will not use them elsewhere in the book, and rest assured that the fact about coends known as the *co-Yoneda lemma* employed in the following proof is a standard and purely formal result.

Proposition 3.79. 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 $(I, \star)$ 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 and q, their Day convolution in $\mathbf{Set}^{\mathbf{Set}}$ is given by the coend

$$p \odot q \cong \int^{(A,B)\in\mathbf{Set}^2} y^{A\star B} \times p(A) \times q(B). \tag{3.80}$$

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 $\mathbf{Set}^{\mathbf{Set}}$ and coends always commute with coproducts (as they are both colimits), we can rewrite (3.80) as

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

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

$$p \odot q \cong \sum_{(i,j)\in p(1)\times q(1)} y^{p[i]\star q[j]}, \tag{3.81}$$

which is in **Poly**. That the Day convolution distributes over coproducts also follows from the fact that products distribute over coproducts in $\mathbf{Set}^{\mathbf{Set}}$ and that coends commute with coproducts; or, alternatively, directly from (3.81).

We observe that (3.81) gives $(y^I, \odot) = (1, \times)$ when $(I, \star) \coloneqq (0, +)$ and $(y^I, \odot) = (y, \otimes)$ when $(I, \star) \coloneqq (1, \times)$. □

Exercise 3.82 (Solution here). There is a monoidal structure on **Set** whose unit is 0 and whose product is given by $(A, B) \mapsto A + AB + B$.

1. Verify that the operation $(A, B) \mapsto A + AB + B$ on **Set** is associative.
2. Verify that 0 is the unit for the above operation.
3. Let $(1, \odot)$ denote the corresponding monoidal structure on **Poly** obtained via

Proposition 3.79. Compute the monoidal product $(y^3 + y) \odot (2y^2 + 2)$. ◊

3.11 Summary and further reading

In this chapter, we introduced the category **Poly**, whose objects are polynomial functors and whose morphisms are the natural transformations between them. We call these natural transformations *dependent lenses*, or *lenses* for short. We also proved our first categorical property of **Poly**: that it has all small coproducts.

The main result of this chapter was a concrete characterization of our dependent lenses between polynomial functors. A dependent lens $f : p \to q$ is characterized by its

- *on-positions function*, a function $f_1 : p(1) \to q(1)$ sending p-positions forward to q-positions; and its
- *on-directions functions*, one for each p-position i denoted $f_i^\sharp : q[f_1 i] \to p[i]$ sending $q[f_1 i]$-directions backward to $p[i]$-directions.

This forward-backward relationship is what makes dependent lenses so well-suited for modeling *interaction protocols*. Given two agents with positions and directions, a dependent lens between them defines an interaction protocol that describes how the position of the first agent determines the position of the second agent and how the direction of the second agent determines the direction of the first. This perspective is exhibited by our corolla and polybox pictures for lenses. We studied examples of lenses between special polynomials: in particular, lenses between monomials are known as *bimorphic lenses* in functional programming literature.

We then unwound our interpretation of natural transformations between polynomials as dependent lenses with on-positions and on-directions functions to describe what happens to these functions when lenses compose. This gave us an accessible way to interpret commutative diagrams in **Poly** that is particularly convenient to express using polyboxes.

Finally, we considered various categorical structures on **Poly**, e.g. that it has all products and coproducts, and that these distribute: $\prod \sum \to \sum \prod$.

$$\sum_{a \in A} p_a \coloneqq \sum_{(a,i) \in \sum_{a \in A} p_a(1)} y^{p_a[i]} \qquad \prod_{a \in A} p_a \coloneqq \sum_{i \in \prod_{a \in A} p_a(1)} y^{\sum_{a \in A} p_a[ia]}$$

$$p_1 + p_2 \coloneqq \sum_{(a,i) \in \{(1,i_1) \mid i_1 \in p_1(1)\} + \{(2,i_2) \mid i_2 \in p_2(1)\}} y^{p_a[i]} \qquad p_1 \times p_2 \coloneqq \sum_{(i_1,i_2) \in p_1(1) \times p_2(1)} y^{p_1[i_1] + p_2[i_2]}$$

We also discussed how one can take any monoidal product $\star$ from **Set** and lift it to a monoidal product $\cdot$ on **Poly**:

$$p_1 \odot p_2 \coloneqq \sum_{i_1, i_2) \in p_1(1) \times p_2(1)} y^{p_1[i_1] \cdot p_2[i_2]}$$

A special case of this is the product structure $\times$ on **Poly**, which emerges from the coproduct structure $+$ on **Set**. The other case of interest is the parallel (or Dirichlet) product structure $\otimes$ on **Poly**, which emerges from the product structure $\times$ on **Set**:

$$p_1 \otimes p_2 \coloneqq \sum_{i_1, i_2) \in p_1(1) \times p_2(1)} y^{p_1[i_1] \times p_2[i_2]}$$

Variants of lenses are studied in compositional game theory [Hed+16; Hed17; Hed18a; Hed18b], in categorical database theory [JRW12], in functional programming and programming language theory [BPV06; OCo11; Abo+16], and in more generalized categorical settings [GJ12; Spi19].

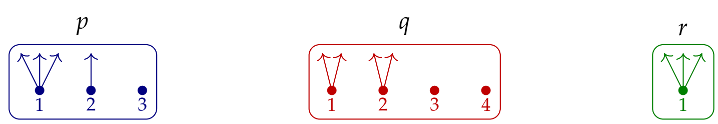
3.12 Exercise solutions

Solution to Exercise 3.2.

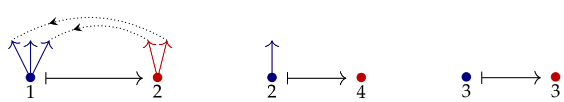
We know that **Poly** is the full subcategory of $\mathbf{Set}^{\mathbf{Set}}$ spanned by polynomial functors, including y^S and q. So $\mathbf{Poly}(y^S, q) = \mathbf{Set}^{\mathbf{Set}}(y^S, q)$. Hence the natural isomorphism $\mathbf{Poly}(y^S, q) \cong q(S)$ follows directly from the Yoneda lemma (Lemma 1.10) with $F \coloneqq q$.

Solution to Exercise 3.12.

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



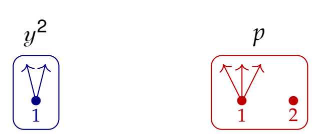
2. Here is one possible lens $p \to q$ (you may have drawn others).



3. As depicted, our lens assigns to the first position of p the second position of q, whose first and second directions are passed back to the third and first directions, respectively, of the first position of p. Then the second position of p is assigned the fourth position of q, which has no directions; effectively, the choice of direction of the second position of p has been canceled. Finally, the third position of p is assigned the third position of q; here neither position has any directions.
4. There cannot be a lens $p \to r$ for the following reason: if we send the third position of p, which has no directions, to the sole position of r, which has 3 directions, then there is no way to pass a choice of one of those 3 directions back to any of the options on the third menu of p, as there are no such options.

Solution to Exercise 3.16.

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



2. When constructing a lens $y^2 \to p$, the unique position of y^2 can be sent to either p-position. If it is sent to the first p-position, then each of the 3 directions in $p[1]$ must be sent to one of the 2

directions of y^2, for a total of $2^3 = 8$ lenses. 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 lens like this. Hence there are $8 + 1 = 9$ lenses $y^2 \to p$.

3. The cardinality of $p(2)$ is $|2^3 + 1| = 9$, which agrees with previous answer, as predicted by the Yoneda lemma.

Solution to Exercise 3.17.

By (3.7), we have for all $p, q \in \mathbf{Poly}$ that

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

1. If $p \coloneqq y^3$ and $q \coloneqq y^4$, then

$$|\mathbf{Poly}(p, q)| = \prod_{i \in 1} |p[i]|^4 = 3^4 = 81.$$

2. If $p \coloneqq y^3 + 1$ and $q \coloneqq y^4$, then

$$|\mathbf{Poly}(p, q)| = \prod_{i \in 2} |p[i]|^4 = 3^4 \cdot 0^4 = 0.$$

3. If $p \coloneqq y^3 + 1$ and $q \coloneqq y^4 + 1$, then

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

4. If $p \coloneqq 4y^3 + 3y^2 + y$ and $q \coloneqq y$, then

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

5. If $p \coloneqq 4y^3$ and $q \coloneqq 3y$, then

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

Solution to Exercise 3.18.

1. By (3.7), it suffices to show that for all $i \in p(1)$ and $j \in q(1)$, we have

$$p[i]^{q[j]} \cong \prod_{b \in q[j]} \sum_{a \in p[i]} 1.$$

Indeed, by (1.30) and Exercise 1.17, we have

$$\begin{aligned}
\prod_{b \in q[j]} \sum_{a \in p[i]} 1 &\cong \sum_{\bar{a} \colon q[j] \to p[i]} \prod_{b \in q[j]} 1 && (1.30) \\
&\cong \sum_{\bar{a} \colon q[j] \to p[i]} 1 && \text{(Exercise 1.17 \#2)} \\
&\cong \mathbf{Set}(q[j], p[i]) && \text{(Exercise 1.17 \#1)} \\
&\cong p[i]^{q[j]}.
\end{aligned}$$

2. By (3.7) and (1.30), we have

$$\begin{aligned}
\mathbf{Poly}(p, q) &\cong \prod_{i \in p(1)} \sum_{j \in q(1)} p[i]^{q[j]} && (3.7) \\
&\cong \sum_{f_1 \colon p(1) \to q(1)} \prod_{i \in p(1)} p[i]^{q[f_1 i]} && (1.30)
\end{aligned}$$

$$\cong \sum_{f_1\colon p(1)\to q(1)} \prod_{j\in q(1)} \prod_{\substack{i\in p(1),\\ f_1 i = j}} p[i]^{q[j]} \tag{$*$}$$

$$\cong \sum_{f_1\colon p(1)\to q(1)} \prod_{j\in q(1)} \mathbf{Set}\left(q[j], \prod_{\substack{i\in p(1),\\ f_1 i = j}} p[i]\right) \qquad \text{(Universal property of products)}$$

where $(*)$ follows from the fact that for any function $f_1\colon p(1)\to q(1)$, its domain $p(1)$ can be written as the disjoint union of preimages $f_1^{-1}(j) = \{i \in p(1) \mid f_1 i = j\}$ for each $j \in q(1)$.

3. To explain how an element of the set

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

corresponds to a lens $p \to q$, 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 sends each p-position to a q-position, as an on-positions function should. Then for each q-position j, each $q[j]$-direction b, and each p-position i that f_1 sends to j, we choose an element of $p[i]$ that $f_i^\sharp\colon q[j]\to p[i]$ assigns to b. As every p-position i is sent to some q-position j by f_1, this completely characterizes $f_i^\sharp$ for every p-position i.

Solution to Exercise 3.21.

We have

$$\begin{aligned}
\mathbf{Poly}\left(\sum_{(i,j)\in\sum_{i\in I} p_i(1)} y^{p_i[j]}, q\right) &\cong \prod_{(i,j)\in\sum_{i\in I} p_i(1)} q(p_i[j]) && (3.7)\\
&\cong \prod_{i\in I}\prod_{j\in p_i(1)} q(p_i[j])\\
&\cong \prod_{i\in I} \mathbf{Poly}(p_i, q). && (3.7)
\end{aligned}$$

Solution to Exercise 3.23.

1. We construct a lens $\dot{p}y \to p$, or a lens
$$\sum_{i\in p(1)}\sum_{a\in p[i]} y^{p[i]} \to \sum_{i\in p(1)} y^{p[i]},$$
as follows. It sends each position $(i,a)\in\sum_{i\in p(1)} p[i]$ of $\dot{p}y$ to its first projection $i\in p(1)$, and it is the identity on directions.
2. There is not always a lens $p \to \dot{p}$: if $p \coloneqq 1$, then $\dot{p} \coloneqq 0$, and there is no lens $1 \to 0$.
3. There is not always a lens $\dot{p} \to p$: take $p \coloneqq y$, so $\dot{p} \coloneqq 1$. A lens $1 \to y$ must have an on-directions function $1 \to 0$, but there is no such function.
4. We show that even when there is a lens $p \to q$, there is not necessarily a lens $\dot{p} \to \dot{q}$. Take $p \coloneqq y$ and $q \coloneqq 1$. Then there is a lens $p \to q$ 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 lens $1 \to 0$.

5. We construct a lens $g\colon [p, y] \otimes p \to \dot{p}$, where $[p, y] \otimes p$ is given by (3.24), as follows. 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 $fi \in p[i]$. Then $\dot{p}[(i, fi)] = p[i] - \{fi\}$ 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 $a \in p[i] - \{fi\}$ to $(i, a) \in p(1) \times p[i]$.
6. We describe a lens $py \to q$ in terms of "unassigned" directions. Observe that py has the same positions as p but has one more direction than p does at each position. Given a position $i \in p(1)$, we denote this extra $(py)[i]$-direction by $*_i$, identifying $(py)[i]$ with $p[i] + \{*_i\}$. So a lens $f\colon py \to q$ sends each p-position i to a q-position j, but every $q[j]$-direction could be sent back to either an original $p[i]$-direction or the extra $(py)[i]$-direction $*_i$. So a lens $py \to q$ is like a lens $p \to q$, except that some of the directions of q may remain "unassigned" to any direction of p, which we signify by assigning them to $*_i$ instead. In other words, a lens $py \to q$ could be interpreted as a lens $p \to q$ whose on-directions functions may only be partially defined.

Solution to Exercise 3.25.

We have

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

which is precisely the set of all directions of p.

Solution to Exercise 3.27.

1. Since $0 \cong 0y$ is a linear polynomial, Example 3.26 tells us that lenses $0 \to p$ can be identified with functions $0 \to p(1)$. There is exactly one function $0 \to p(1)$, so there is exactly one lens $0 \to p$.
2. Since $y \cong 1y$ is a linear polynomial, Example 3.26 tells us that lenses $y \to p$ can be identified with functions $1 \to p(1)$, which in turn can be identified with elements of $p(1)$.

Solution to Exercise 3.28.

A lens $f\colon p \to Iy$ consists of an on-positions function $f_1\colon p(1) \to I$ and, for each $j \in p(1)$, an on-directions function $f^\sharp_j\colon 1 \to p[j]$. Equivalently, this is a function $p(1) \to I$ and a choice of direction at every position of p, i.e. a dependent function $(j \in p(1)) \to p[j]$.

Solution to Exercise 3.30.

1. Since 1 is a constant, Example 3.29 tells us that lenses $p \to 1$ can be identified with functions $p(1) \to 1$. There is exactly one function $p(1) \to 1$, so there is exactly one lens $p \to 1$.
2. Since 0 is a constant, Example 3.29 tells us that lenses $p \to 0$ can be identified with functions $p(1) \to 0$, which only exist when $p(1) = 0$. The only polynomial with an empty position-set is 0 itself, and there is a unique function from the set 0 to itself, so there is a unique lens from the constant polynomial 0 to itself as well. If p is not the constant 0, then there are no lenses $p \to 0$.

Solution to Exercise 3.31.

1. A lens $f\colon I \to p$ consists of an on-positions function $f_1\colon I \to p(1)$ and, for each $i \in I$, an on-directions function $f^\sharp_i\colon p[i] \to 0$. There is exactly one such on-directions function when $p[i] = 0$ and no such on-directions function otherwise. It follows that a lens $f\colon I \to p$ can be identified with a function $f_1\colon I \to p(1)$ whose image is contained in the set of p-positions with no directions. By Exercise 2.9, this set of p-positions can be identified with the set $p(0)$ (the *constant* term of p); so a lens $I \to p$ is equivalent to a function $I \to p(0)$.
2. From the previous part, a lens $1 \to p$ may be identified with a function $1 \to p(0)$ and thus an element of $p(0)$.

Solution to Exercise 3.40.

The functor Γ is defined as the hom-functor $\mathbf{Poly}(-, y)\colon \mathbf{Poly} \to \mathbf{Set}^{\mathrm{op}}$, which exhibits the universal property of colimits by sending colimits in $\mathbf{Poly}$ to limits in $\mathbf{Set}$. Hence Proposition 3.39 follows from Proposition 3.3. More explicitly, $\Gamma(0) = \mathbf{Poly}(0, y) \cong 1$ 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 3.48.

Fix $i \in p(1)$. Since the $p[i]$-component $(\mathrm{id}_p)_{p[i]}$ of id_p is the identity function on $p(p[i])$, by Corollary 3.47,

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

Hence the on-positions function $(\mathrm{id}_p)_1 \colon p(1) \to p(1)$ maps every $i \in p(1)$ to itself, so it is an identity function; and each on-directions function $(\mathrm{id}_p)^\sharp_i \colon p[i] \to p[i]$ is equal to $\mathrm{id}_{p[i]}$.

Solution to Exercise 3.49.

Fix $i \in p(1)$. By Proposition 3.44 and Corollary 3.47,

$$\begin{aligned}
(h_1 i, h^\sharp_i) &= h_{p[i]}(i, \mathrm{id}_{p[i]}) && \text{(Corollary 3.47)}\\
&= g_{p[i]}(f_{p[i]}(i, \mathrm{id}_{p[i]})) && (h = f \mathbin{⨾} g)\\
&= g_{p[i]}(f_1 i, f^\sharp_i) && \text{(Corollary 3.47)}\\
&= (g_1 f_1 i, g^\sharp_{f_1 i} \mathbin{⨾} f^\sharp_i). && \text{(Proposition 3.44)}
\end{aligned}$$

Solution to Exercise 3.51.

We provide lenses $\iota \colon p \to p+q$ 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. To show that $p+q$ equipped with ι and κ satisfies the universal property of the coproduct, we apply Proposition 3.50. In order for (3.52) to commute, it must commute on positions—that is, the following diagram of sets must commute:

$$\begin{tikzcd}
p(1) \arrow[r, "\iota_1"] \arrow[dr, "f_1"'] & p(1) + q(1) \arrow[d, dashed, "h_1"] & q(1) \arrow[l, "\kappa_1"'] \arrow[dl, "g_1"] \\
& r(1) &
\end{tikzcd} \tag{3.83}$$

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 (3.83) 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 $g_1 j$.

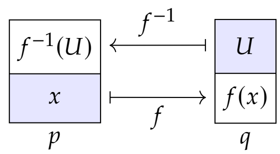
Moreover, if (3.52) is to commute on directions, then for every $i \in p(1)$ and $j \in q(1)$, the following diagrams of sets must commute:

$$\begin{tikzcd}
p[i] & (p+q)[(1,i)] \arrow[l, "\iota^\sharp_i"'] \\
& r[f_1 i] \arrow[ul, "f^\sharp_i"] \arrow[u, dashed, "h^\sharp_{(1,i)}"']
\end{tikzcd}
\qquad
\begin{tikzcd}
(p+q)[(2,j)] \arrow[r, "\kappa^\sharp_j"] & q[j] \\
r[g_1 j] \arrow[u, dashed, "h^\sharp_{(2,j)}"] \arrow[ur, "g^\sharp_j"']
\end{tikzcd} \tag{3.84}$$

But $(p+q)[(1,i)] \cong p[i]$ and $\iota^\sharp_i$ is the identity, so we must have $h^\sharp_{(1,i)} = f^\sharp_i$. Similarly, $(p+q)[(2,j)] \cong q[j]$ and $\kappa^\sharp_j$ is the identity, so we must have $h^\sharp_{(2,j)} = g^\sharp_j$. 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 (3.83) commutes, and that we can define h on directions such that the diagrams in (3.84) commute. As the commutativity of the diagrams in (3.83) and (3.84) together imply the commutativity of (3.52), it follows that there exists h for which (3.52) commutes.

Solution to Exercise 3.53.

1. Given a continuous map $f\colon X \to Y$, we define a lens $p_f\colon p_X \to p_Y$ as follows. The on-positions function is just f; then for each p_X-position $x \in X$, the on-directions function $(p_f)^\sharp_x\colon p_Y[f(x)] \to p_X[x]$ sends each open neighborhood U of $f(x)$ to $f^{-1}(U)$, which we know is an open neighborhood of x because f is continuous. The polybox picture for p_f is as follows:



2. 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 for $X \in \mathbf{Top}$ that the lens p_{id_X} is an identity. By #1, the on-positions function of p_{id_X} is id_X, and for each $x \in X$ the on-directions function $(p_f)^\sharp_x\colon p_X[x] \to p_X[x]$ sends $U \in p_X[x]$ to $(\mathrm{id}_X)^{-1}(U) = U$. Hence p_{id_X} is the identity on both positions and directions; it follows from Exercise 3.48 that p_{id_X} is the identity lens.
We now show for $X, Y, Z \in \mathbf{Top}$ and continuous maps $f\colon X \to Y$ and $g\colon Y \to Z$ that $p_f \mathbin{⨾} p_g = p_{f ⨾ g}$. By #1 and Exercise 3.49, the on-positions functions of both $p_f \mathbin{⨾} p_g$ and $p_{f⨾g}$ are equal to $f \mathbin{⨾} g$, so it suffices to show for each $x \in X$ that
$$(p_{f⨾g})^\sharp_x = (p_g)^\sharp_{f(x)} \mathbin{⨾} (p_f)^\sharp_x.$$
By #1, the left hand side sends each $U \in p_Z[g(f(x))]$ to $(f \mathbin{⨾} g)^{-1}(U)$, while the right hand side sends U to $f^{-1}(g^{-1}(U))$; by elementary set theory, these sets are equal.
3. The functor is not full. Consider the spaces $X \coloneqq 2$ with the indiscrete topology (i.e. the only open sets are the empty set and X) and $Y \coloneqq 2$ with the discrete topology (i.e. all subsets are open). Then $p_X \cong 2y$ (each point in X has exactly one open neighborhood: the entire space X) and $p_Y \cong 2y^2$ (each point in Y has exactly two open neighborhoods: a singleton set and Y itself), so our functor induces a map from the set of continuous functions $X \to Y$ to the set of lenses $2y \to 2y^2$. We claim this map 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: a singleton subset of Y is open, but its preimage under f is a singleton subset of X and therefore not open. So our functor sends no continuous function $X \to Y$ to h and therefore is not full. The functor is, however, faithful: given spaces X and Y and continuous function $f\colon X \to Y$, we can uniquely recover f from p_f by taking its on-positions function $(p_f)_1 = f$.

Solution to Exercise 3.59.

Given $q \in \mathbf{Poly}$ and $p_i \in \mathbf{Poly}$ for each $i \in I$ for some set I, we use (3.7) to verify that
$$\begin{aligned}
\mathbf{Poly}\left(q, \prod_{i\in I} p_i\right) &\cong \prod_{k \in q(1)} \left(\prod_{i\in I} p_i\right)(q[k]) && (3.7)\\
&\cong \prod_{k\in q(1)} \prod_{i \in I} p_i(q[k]) \\
&\cong \prod_{i\in I} \prod_{k \in q(1)} p_i(q[k]) \\
&\cong \prod_{i \in I} \mathbf{Poly}(q, p_i). && (3.7)
\end{aligned}$$

Solution to Exercise 3.60.

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 (3.57) as follows:

$$\begin{aligned}
\prod_{i\in 3} p_i &\cong \sum_{\bar{j}\in\prod_{i\in 3} p_i(1)} y^{\sum_{i\in 3} p_i[\bar{j}(i)]} && (3.57)\\
&\cong \sum_{\bar{j}\colon (i\in 3)\to p_i(1)} y^{p_1[\bar{j}(1)]+p_2[\bar{j}(2)]+p_3[\bar{j}(3)]}\\
&\cong y^{p_1[1]+p_2[1]+p_3[1]} + y^{p_1[1]+p_2[2]+p_3[1]} + y^{p_1[1]+p_2[3]+p_3[1]}\\
&+ y^{p_1[2]+p_2[1]+p_3[1]} + y^{p_1[2]+p_2[2]+p_3[1]} + y^{p_1[2]+p_2[3]+p_3[1]}\\
&\cong y^{1+1+2} + y^{1+0+2} + y^{1+0+2}\\
&+ y^{0+1+2} + y^{0+0+2} + y^{0+0+2}\\
&\cong y^4 + 3y^3 + 2y^2,
\end{aligned}$$

as we might expect from standard polynomial multiplication.

Solution to Exercise 3.63.

1. We compute the product using (3.61):

$$\begin{aligned}
B_1y^{A_1} \times B_2y^{A_2} &\cong \left(\sum_{i\in B_1} y^{A_1}\right) \times \left(\sum_{j\in B_2} y^{A_2}\right)\\
&\cong \sum_{i\in B_1}\sum_{j\in B_2} y^{A_1+A_2}\\
&\cong B_1B_2y^{A_1+A_2}.
\end{aligned}$$

2. We expand the product by applying (1.28), with $I_1 \coloneqq I$ and $I_2 \coloneqq J$:

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

where the last isomorphism follows from #1.

Solution to Exercise 3.64.

We wish to show that, for $p, q \in \mathbf{Poly}$, the polynomial pq along with the lenses $\pi\colon pq \to p$ and $\varphi\colon pq \to q$ 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 lenses $f\colon r \to p$ and $g\colon r \to q$, there exists a unique lens $h\colon r \to pq$ for which the following diagram commutes:

$$\begin{tikzcd}
r \arrow[r, "g"] \arrow[d, "f"'] \arrow[dr, dashed, "h"] & q \\
p & pq. \arrow[l, "\pi"] \arrow[u, "\varphi"']
\end{tikzcd} \tag{3.85}$$

We apply Proposition 3.50. In order for (3.85) to commute, it must commute on positions—that is, the following diagram of sets must commute:

$$\begin{tikzcd}
r(1) \arrow[r, "g_1"] \arrow[d, "f_1"'] \arrow[dr, dashed, "h_1"] & q(1) \\
p(1) & (pq)(1). \arrow[l, "\pi_1"] \arrow[u, "\varphi_1"']
\end{tikzcd} \tag{3.86}$$

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 (3.86) 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 (3.52) is to commute on directions, then for every $k \in r(1)$, the following diagram of sets must commute:

$$\begin{tikzcd}
r[k] & & q[g_1 k] \arrow[ll, "g_k^\sharp"'] \arrow[d, "\varphi^\sharp_{(f_1 k, g_1 k)}"] \\
p[f_1 k] \arrow[u, "f_k^\sharp"] \arrow[rr, "\pi^\sharp_{(f_1 k, g_1 k)}"'] & & (pq)[(f_1 k, g_1 k)]. \arrow[ull, dashed, "h_k^\sharp"']
\end{tikzcd} \tag{3.87}$$

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 $\varphi^\sharp_{(f_1 k, g_1 k)}$ form the coproduct of $p[f_1 k]$ and $q[g_1 k]$ in **Set**, there exists a unique $h_k^\sharp$ for which (3.87) 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 (3.86) commutes, and that we can define h on directions such that (3.87) commutes. As the commutativity of (3.86) and (3.87) together imply the commutativity of (3.85), it follows that there exists h for which (3.85) commutes.

Solution to Exercise 3.67.

1. We compute the parallel product using (3.66):

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

2. We expand the parallel product as follows:

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

Solution to Exercise 3.68.

1. By Exercise 3.67 #1, we have $A \otimes B \cong A y^0 \otimes B y^0 \cong AB y^0 \cong AB$.
2. By Exercise 3.67 #1, we have $Ay \otimes By \cong Ay^1 \otimes By^1 \cong ABy^1 \cong ABy$.
3. By (3.66),

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

Solution to Exercise 3.69.

1. We compute $p \times q$ using Exercise 3.63 #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 3.67 #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 ordinary laws of arithmetic:

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

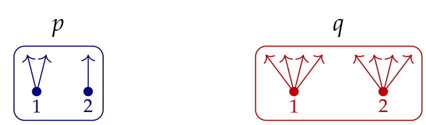
4. We describe the connection between the last two parts as follows. Given a polynomial p, we let $d(p)$ denote the Dirichlet series $\sum_{i\in p(1)} |p[i]|^y$. Then by (3.66),

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

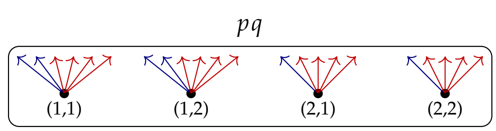
The last two parts are simply an example of this identity for a specific choice of p and q.

Solution to Exercise 3.71.

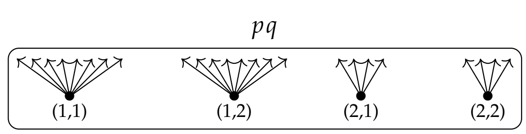
1. Here are p and q drawn as corolla forests:



2. Here is pq drawn as a corolla forest:



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



Solution to Exercise 3.72.

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

$$\begin{aligned} p\otimes y &\cong \sum_{i\in p(1)} \sum_{j\in 1} y^{p[i]\times 1} && (3.66) \\ &\cong \sum_{i\in p(1)} y^{p[i]} \cong p. \end{aligned}$$

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

$$(p\otimes q)\otimes r \cong \left(\sum_{i\in p(1)} \sum_{j\in q(1)} y^{p[i]\times q[j]}\right)\otimes r \qquad (3.66)$$

$$
\begin{aligned}
&\cong \sum_{i \in p(1)} \sum_{j \in q(1)} \left(\sum_{k \in r(1)} y^{(p[i] \times q[j]) \times r[k]} \right) && (3.66)\\
&\cong \sum_{i \in p(1)} \left(\sum_{j \in q(1)} \sum_{k \in r(1)} y^{p[i] \times (q[j] \times r[k])} \right) && (\text{Associativity of } \textstyle\sum \text{ and } \times)\\
&\cong p \otimes \left(\sum_{j \in q(1)} \sum_{k \in r(1)} y^{q[j] \times r[k]} \right) && (3.66)\\
&\cong p \otimes (q \otimes r). && (3.66)
\end{aligned}
$$

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

$$
\begin{aligned}
p \otimes q &\cong \sum_{i \in p(1)} \sum_{j \in q(1)} y^{p[i] \times q[j]} && (3.66)\\
&\cong \sum_{j \in q(1)} \sum_{i \in p(1)} y^{q[j] \times p[i]} && (\text{Commutativity of } \textstyle\sum \text{ and } \times)\\
&\cong q \otimes p. && (3.66)
\end{aligned}
$$

Solution to Exercise 3.74.

1. We compute $(3y^5 + 6y^2) \otimes 4$ using Exercise 3.67 #2 and the fact that $4 = 4y^0$:

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

2. Given a polynomial p and a set J viewed as a constant polynomial, we have

$$
\begin{aligned}
p \otimes J &\cong \left(\sum_{i \in p(1)} y^{p[i]} \right) \otimes \left(\sum_{j \in J} y^0 \right)\\
&\cong \sum_{i \in p(1)} \sum_{j \in J} y^{p[i] \times 0}\\
&\cong p(1)Jy^0\\
&\cong p(1)J,
\end{aligned}
$$

itself a constant polynomial; so the class of constant polynomials is indeed a $\otimes$-ideal.

Solution to Exercise 3.75.

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 3.68 #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 3.68 #2 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 3.67 #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 quadratic polynomials is not closed under $\otimes$, for even though $y^2 \cong 1y^2 + 0y + 0$ is a quadratic polynomial, Exercise 3.67 #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 3.67 #1 and does contain y, as $y \cong 1y^1$.
6. The set $\{Sy^S \mid S \in \mathbf{Set}\}$ is closed under $\otimes$, for Exercise 3.67 #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 3.68 #3 and the fact that the product of two finite sets is itself finite. The set contains y, as $y(1) \cong 1$ is finite.

Solution to Exercise 3.76.

The smallest class of polynomials that is closed under $\otimes$ and contains y is just $\{y\}$. This is because by Exercise 3.67 #1, we have $y \otimes y \cong y$.

Solution to Exercise 3.77.

We show that $(p_1 + p_2) \otimes q \cong (p_1 \otimes q) + (p_2 \otimes q)$ using (3.66):
$$\begin{aligned}(p_1 + p_2) \otimes q &\cong \sum_{k \in 2} \sum_{i \in p_k(1)} \sum_{j \in q(1)} y^{p_k[i] \times q[j]} \\ &\cong \sum_{i \in p_1(1)} \sum_{j \in q(1)} y^{p_1[i] \times q[j]} + \sum_{i \in p_2(1)} \sum_{j \in q(1)} y^{p_2[i] \times q[j]} \\ &\cong (p_1 \otimes q) + (p_2 \otimes q).\end{aligned}$$

Solution to Exercise 3.82.

1. To show that the operation $(A, B) \mapsto A + AB + B$ on $\mathbf{Set}$ is associative, 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 the unit for this operation, observe that
$$(A, 0) \mapsto A + A0 + 0 \cong A$$
and
$$(0, B) \mapsto 0 + 0B + B \cong B.$$
3. Taking $A \star B \coloneqq A + AB + B$ in Proposition 3.79 to obtain a monoidal product $\odot$ on $\mathbf{Poly}$, we can use (3.81) to compute that
$$\begin{aligned}(y^3 + y) \odot (2y^2 + 2) &\cong (y^3 + y^1) \odot (y^2 + y^2 + y^0 + y^0) \\ &\cong y^{3\star 2} + y^{3\star 2} + y^{3\star 0} + y^{3\star 0} + y^{1\star 2} + y^{1\star 2} + y^{1\star 0} + y^{1\star 0} \\ &\cong 2y^{11} + 2y^3 + 2y^5 + 2y^1.\end{aligned}$$

Chapter 4

Dynamical systems as dependent lenses

One of the main goals of this book is to use dependent lenses in **Poly** to model dynamical systems and automata. In this chapter, we will begin to see how to do this through an array of examples.

4.1 Moore machines

We start with our simplest example of a dynamical system: a deterministic state machine with a fixed range of states, inputs, and outputs. At any point in time, this machine will inhabit one of its possible states and return output according to that current state. It can also update its current state according to the input it receives.

Definition 4.1 (Moore machine)**.** A *Moore machine* consists of the following data: three sets,

- a set S, called the *state-set*, whose elements are *states*;
- a set I, called the *position-set* (or *output-set*), whose elements are *positions* (or *outputs*);
- a set A, called the *direction-set* (or *input-set*), whose elements are *directions* (or *inputs*);

and two functions,

- return: $S \to I$;
- update: $S \times A \to S$.

To emphasize the role that the three sets play, we can specify that this is an (A, I)-*Moore machine with states* S.

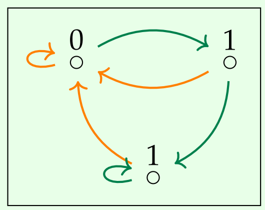
The input/output terminology is standard, while the position/direction terminology is our own: we will soon see how the positions and directions of a Moore machine relate to that of a polynomial.

We should interpret an (A, I)-Moore machine as follows. At any time, the machine inhabits one of the states in its state-set S. Say its current state is $s \in S$. We can ask the machine to perform one of the following two tasks.

- We can ask the machine to *return its position*: it should then produce the position $\text{return}(s) \in I$.
- We can feed the machine one of its *directions* $a \in A$ and ask it to *update its state*: it should then replace its current state with the new state $\text{update}(s, a) \in S$. Note that the new state depends not only on the direction the machine receives but also on the state the machine inhabits when it receives that direction.

We may visualize a Moore machine with a *transition diagram* as follows.

Example 4.2 (A Moore machine's transition diagram). Given $A := \{\text{orange}, \text{green}\}$ and $I := \{0, 1\}$, we can draw a transition diagram for an (A, I)-Moore machine with $S := 3$ states as follows:



(4.3)

Each state is labeled by the position it returns according to the machine's return function. Additionally, each state has two outgoing arrows, one orange and one green, corresponding to the two possible directions. The targets of the arrows indicate the updated state according to the machine's update function.

Say the machine starts at the bottom state. By feeding it a sequence of directions—say (orange, orange, green, orange, ...)—we can send the machine through its states via its update function and return the position at each state:

1. Starting at the bottom state, the machine returns the position 1.
2. Following the orange arrow from the bottom state, the machine updates its state to the left state.
3. At the left state, the machine returns the position 0.
4. Following the orange arrow from the left state, the machine updates its state to—once again—the left state.
5. At the left state, the machine returns the position 0.
6. Following the green arrow from the left state, the machine updates its state to the right state.
7. At the right state, the machine returns the position 1.
8. Following the orange arrow from the right state, the machine updates its state to the left state.
9. At the left state, the machine returns the position 0.

...

In summary, starting from the bottom state, this Moore machine sends the sequence (orange, orange, green, orange, . . .) of directions in A to the sequence $(1, 0, 0, 1, 0, \ldots)$ of positions in I.

In general, given an initial state $s_0 \in S$, an (A, I)-Moore machine with states S sends every sequence $(a_1, a_2, a_3, \ldots)$ of directions in A to a sequence $(i_0, i_1, i_2, i_3, \ldots)$ of positions in I, defined inductively as follows, via an intermediary sequence $(s_0, s_1, s_2, s_3, \ldots)$ of states in S:

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

for all $k \in \mathbb{N}$. We will see that **Poly** gives us a more concise way to express this in Example 8.52.

Comparing Definition 4.1 with Example 3.41, we find that an (A, I)-Moore machine with states S is precisely a lens between monomials $\varphi\colon Sy^S \to Iy^A$ with on-positions function $\varphi_1 \coloneqq \text{return}\colon S \to I$ and on-directions map $\varphi^\sharp \coloneqq S \times A \to S$. The positions and directions of the Moore machine are the positions and directions of the codomain of the corresponding lens, while the domain of the lens has the states of the Moore machine as both its positions and its directions. So we can repackage Definition 4.1 as follows.

Definition 4.4 (Moore machine, version 2)**.** For $S, I, A \in$ **Set**, an (A, I)-*Moore machine with states* S is a lens

$$\varphi\colon Sy^S \to Iy^A$$

in **Poly**. We call

- the domain monomial Sy^S the machine's *state system*: its position-set (equivalently, its direction-set) is the machine's *state-set*, and its positions (equivalently, its directions) are the machine's *states*;
- the codomain monomial Iy^A the machine's *interface*: its position-set and direction-set are the machine's *position-set* and *direction-set*, and its positions and directions are the machine's *positions* and *directions*;
- the on-positions function $\varphi_1\colon S \to I$ the machine's *return function*;
- the on-directions map $\varphi^\sharp\colon S \times A \to S$ the machine's *update function*.

We call the codomain of a Moore machine its *interface* because it encodes how an outsider interacts with the machine: an outsider observes the positions of the interface that the machine returns and feeds the directions of the interface to the machine to update it. Rather than directly observing and altering the machine's states, an outsider must interact with the machine via its interface.

Exercise 4.5 (Solution here)*.* In this exercise, we will write the Moore machine from Example 4.2 as a lens φ between monomials.

1. What is the machine's state system, the domain of φ?

2. What is the machine's interface, the codomain of φ?

Call the left state L, the right state R, and the bottom state B.

3. What is the machine's return function, the on-positions function of φ?
4. What is the machine's update function, the on-directions map of φ?
5. Draw the first two steps listed in Example 4.2 of the machine's operation (starting at the bottom state and receiving the direction orange) using polyboxes. ◊

Here are some more examples of Moore machines.

Example 4.6 (Counter). There is a $(1, \mathbb{N})$-Moore machine with states $\mathbb{N}$ that, with initial state $0 \in \mathbb{N}$, returns the sequence of natural numbers $(0, 1, 2, 3, \ldots)$. The machine is given by the lens $\mathbb{N}y^{\mathbb{N}} \to \mathbb{N}y$ whose on-positions function is the identity $\mathbb{N} \to \mathbb{N}$ and whose on-directions map $\mathbb{N} \times 1 \cong \mathbb{N} \to \mathbb{N}$ sends $n \mapsto n + 1$. Here it is in polyboxes (recall that the shaded direction box indicates that the direction-set is a singleton, i.e. there is no choice to be made in filling it in):

$$\begin{array}{r|c|c|c|l}
\mathbb{N} & n+1 & \longleftarrow\!\!\!\!| & \blacksquare & \\
\mathbb{N} & n & \longmapsto & n & \mathbb{N}
\end{array}$$

The picture tells us that if the current state (the left position box) is $n \in \mathbb{N}$, the next state (the left direction box) is $n + 1 \in \mathbb{N}$. Since the machine just returns its current state as a position, the sequence of positions returned will always be an increasing sequence of consecutive natural numbers starting at the initial state.

Example 4.7 (Moving in the plane). Let us construct a Moore machine with positions in $\mathbb{R}^2$, which we may think of as locations in the coordinate plane, and directions in $[0, \infty) \times [0, 2\pi)$, which we may think of as commands to move a certain distance $r \in [0, \infty)$ at a certain angle $\theta \in [0, 2\pi)$. We will let the machine's state-set be $\mathbb{R}^2$ as well, so the machine is a lens

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

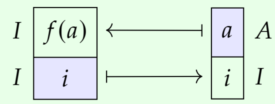
We can define such a lens using polyboxes:

$$\begin{array}{r|c|c|c|l}
\mathbb{R}^2 & (x + r\cos\theta, y + r\sin\theta) & \longleftarrow\!\!\!\!| & (r, \theta) & [0, \infty) \times [0, 2\pi) \\
\mathbb{R}^2 & (x, y) & \longmapsto & (x, y) & \mathbb{R}^2
\end{array}$$

Exercise 4.8 (Solution here). Explain in words what the Moore machine in Example 4.7 does. ◊

Example 4.9 (Functions as memoryless Moore machines). Given a function $f\colon A \to I$, there is a corresponding (A, I)-Moore machine with states I that takes in an element of A and returns the element of I obtained by applying f.

It is given by the lens $Iy^I \to Iy^A$ defined as follows:



That is, this lens is the identity on positions, returning the state directly as its position, and on directions it is the function $I \times A \xrightarrow{\pi_2} A \xrightarrow{f} I$, which ignores the current state and applies f to the direction received to compute the new state.

If the machine starts in state i_0 and is given a sequence of directions $(a_1, a_2, \ldots)$ from A, the machine will return the positions $(i_0, f(a_1), f(a_2), \ldots)$. We say this machine is *memoryless*, because at no point does the state of the machine actually depend on any previous states; instead, its state depends only on the last direction it received.

Exercise 4.10 (Solution here). Suppose we have a function $f\colon A \times I \to I$.

1. Find a corresponding (A, I)-Moore machine $Iy^I \to Iy^A$. You may draw it out in polyboxes.
2. Would you say the machine is memoryless? ◊

Exercise 4.11 (Solution here). Find $A, I \in \mathbf{Set}$ such that the following can be identified with a lens $Sy^S \to Iy^A$, and explain in words what the corresponding (A, I)-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 describes how to transfer 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 $S' \subseteq S$. ◊

The previous examples of Moore machines mostly had identities as return functions. In the following exercises, we will build examples of Moore machines that do not return their entire states as positions.

Exercise 4.12 (Robot with health; solution here). Think of the Moore machine in Example 4.7 as a robot and modify it as follows.

Add to its state a "health meter," which takes a real value between 0 and 1 representing the robot's health. Have the robot lose half its health each time it moves to a location whose x-coordinate is negative. Do not return the robot's health; instead, use its health h as a multiplier, allowing it to move a distance of hr given an input of r. ◊

Exercise 4.13 (Tape of a Turing machine; solution here). A Turing machine has a tape consisting of a cell for each integer. Each cell bears a value $v \in V := \{0, 1, -\}$, and one of the cells $c \in \mathbb{Z}$ is distinguished as the "current" cell. So 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 of the current cell, an element of V; and by changing the value of the current cell before moving left (i.e. replacing the current cell $c \in \mathbb{Z}$ with the new cell $c - 1$) or right (i.e. replacing c with $c + 1$). Hence the tape's position-set is V and its direction-set is $V \times \{\texttt{left}, \texttt{right}\}$.

1. If we model the tape as a Moore machine $t \colon Sy^S \to Iy^A$, what are S, I, and A?
2. Write down the specific t that makes it act like a tape as specified above. ◊

Exercise 4.14 (File-reader; solution here). Say that a *file* of length n is a function $f \colon \mathsf{n} \to \mathsf{ascii}$, 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 $f(i) \in \mathsf{ascii}$ as the *character* at entry i.

Given a file f, design a file-reading Moore machine whose position-set is $\mathsf{ascii} + \{\texttt{done}\}$ and whose direction-set is

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

Given a direction (s, t), the file-reader should go to entry s in the file and return the character at that entry. If the given direction is instead `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 return `done` until it receives another (s, t) pair. ◊

While Exercise 4.14 gives us a functioning file-reader, it is rather awkward that we are still able to give the direction `continue` even when the position is `done`, or provide a new range of entries before the file-reader has finished reading from the previous range. In Section 4.2, we will introduce a generalization of Moore machines to handle cases like these, where the array of directions the machine can receive changes depending on its current position. In particular, we will be able to let the file-reader "close its port," so that it cannot receive signals while it is busy reading, but open its port once it is `done`; see Example 4.28.

4.1.1 Deterministic state automata

The diagram in Example 4.2 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 distinguished initial state.

Definition 4.15 (Deterministic state automaton, language)**.** A *deterministic state automaton* consists of

- a set S of *states*;
- a set A of *symbols*;
- an *update function* $u\colon S \times A \to S$;
- an *initial state* $s_0 \in S$;
- a subset $F \subseteq S$ of *accept states*.

Let
$$\mathsf{List}(A) = \sum_{n \in \mathbb{N}} A^{\mathsf{n}}$$
denote the set of finite sequences $(a_1, \ldots, a_n)$ of symbols in A; we call such a sequence a *word*. We say that the automaton *accepts* the word $(a_1, \ldots, a_n)$ if starting at the initial state and following the symbols in the word leads us to an accept state—or, more formally, if the sequence $(s_0, s_1, \ldots, s_n)$ defined inductively by
$$s_{k+1} \coloneqq u(s_k, a_{k+1})$$
is such that s_n is an accept state: $s_n \in F$.

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

Remark 4.16. 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 can be cumbersome to work with. In Example 8.51, we will give a more compact way of describing whether an automaton accepts a word and specifying the language the automaton recognizes. Better yet, we will find that this alternative formulation arises naturally from the theory of **Poly**.

Proposition 4.17. A deterministic state automaton with a set of states S and a set of symbols A can be identified with a pair of lenses
$$y \to Sy^S \to \mathsf{2}y^A.$$

Proof. By Exercise 3.27, a lens $y \to Sy^S$ can be identified with an initial state $s_0 \in S$. Then a lens $Sy^S \to \mathsf{2}y^A$ consists of a return 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 $u\colon S \times A \to S$. $\square$

In other words, we can think of a deterministic state automaton as a Moore machine with position set $\mathsf{2}$ along with a distinguished initial state; the Moore machine has the same states and update function as the automaton and the automaton's symbols as its directions.

Now imagine if we wanted to construct a version of this automaton that stops reading symbols (i.e. directions) whenever the machine enters an accept state (i.e. returns one position instead of the other). To do this would require a machine whose set of possible directions is dependent on its current position. Instead of an update function $u\colon S \times A \to S$, we would need an update function that takes a direction $a \in A$ if the state $s \in S$ is *not* an accept state (say, if $f(s) = 1$) but takes a direction in 0 (i.e. no direction) if the state s *is* an accept state (if $f(s) = 2$). So there would be one update function $u_s\colon A \to S$ if $f(s) = 1$ and a different update function $u_s\colon 0 \to S$ 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.

4.2 Dependent dynamical systems

Each of our Moore machines above has a monomial Iy^A as an interface. Every representable summand of such an interface has the same representing set A, so the set of directions that can be fed into the machine is always A. But by replacing Iy^A with an arbitrary polynomial p, which may have a different direction-set at each position, we can model a broader class of machines.

Definition 4.18 (Dependent dynamical system)**.** A *dependent dynamical system* (or a *dependent Moore machine*, or simply a *dynamical system*) is a lens

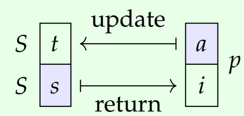
$$\varphi\colon Sy^S \to p$$

for some $S \in \mathbf{Set}$ and $p \in \mathbf{Poly}$. We call

- the domain monomial Sy^S the machine's *state system*—its position-set (equivalently, its direction-set) is the machine's *state-set*, and its positions (equivalently, its directions) are the machine's *states*;
- the codomain polynomial p the machine's *interface*—its position-set and direction-sets are the machine's *position-set* and *direction-sets*, and its positions and directions are the machine's *positions* and *directions*;
- the on-positions function $\varphi_1\colon S \to p(1)$ the machine's *return function*;
- the on-directions map $\varphi^\sharp\colon p[\varphi_1(-)] \to S$ the machine's *update map*, and the on-directions function $\varphi^\sharp_s\colon p[\varphi_1 s] \to S$ at $s \in S$ the machine's *update function* at s.

Example 4.19 (Dynamical systems as polyboxes). We can express a dynamical system

$\varphi \colon Sy^S \to p$ in polyboxes as



We can visualize φ as a channel between the internal state system on the left and the external interface on the right. The state system enters its current state $s \in S$ into the left position box, and the return function converts this state to a position $i \in p(1)$ of the interface. At i, the interface has a direction-set $p[i]$; an interacting agent selects one of these directions $a \in p[i]$ to enter into the right direction box. Finally, the update map uses the current state s and the position i to fill the left direction box with the new state $t \in S$. Then the process repeats with t in place of s.

Remark 4.20. It may seem limiting that the set of possible directions a dependent dynamical system can receive should depend on the current *position* rather than the current *state*; but this makes sense philosophically if we accept that the system's interface should capture *everything* about how it interacts with the outside world. In particular, the system's position should capture everything an external observer could possibly perceive about the system, while the direction-set should capture all the ways in which an external agent can choose to interact with the system. But if the set of directions available to an external agent *changes*, the external agent should be able to detect this fact—the system's position must have changed as well! On the other hand, if the internal state changes, but the external position remains the same, the agent wouldn't see any difference—they wouldn't know to interact with the system any differently, so the directions available to them would have to stay the same, too.

Here are some examples of dependent dynamical systems. We begin by finishing the example at the end of the last section.

Example 4.21 (Halting deterministic state automata). Recall deterministic state automata from Definition 4.15. Say we want such an automaton to halt after reaching an accept state and read no more symbols. Then rather than a lens $Sy^S \to 2y^A$, we could use a lens

$$\varphi \colon Sy^S \to y^A + 1 \cong \{\texttt{reject}\}y^A + \{\texttt{accept}\}.$$

To give such a lens, we first need to provide a return function $\varphi_1 \colon S \to \{\texttt{reject}, \texttt{accept}\}$. We let φ send the accept states to `accept` and every other state to `reject`.

If we reach an accept state, we want the machine to halt. So at the position `accept`, corresponding to the summand 1, there are no directions available. This makes the update function $\varphi^\sharp_s$ vacuous when $\varphi_1 s = \texttt{accept}$.

On the other hand, when $\varphi_1 s = \texttt{reject}$, the update functions $\varphi^\sharp_s \colon A \to S$ specify how the machine updates its state for each direction in A if the current state is s. This corresponds to the automaton's update function.

When equipped with an initial state $s_0 \in S$ specified by a lens $y \to Sy^S$, we call these dependent dynamical systems *halting deterministic state automata*. Given a 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 reaching an accept state any earlier*—or, more formally, if the sequence $(s_0, s_1, \ldots, s_n)$ defined inductively by

$$s_{k+1} \coloneqq \varphi^\sharp_{s_k} a_{k+1}$$

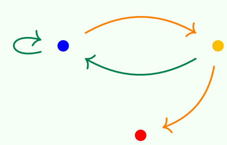
is such that s_n is the sequence's first accept state:

$$\varphi_1 s_k = \begin{cases} \texttt{reject} & \text{if } k < n \\ \texttt{accept} & \text{if } k = n \end{cases}$$

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

Remark 4.22. Again, the conditions for when such an automaton accepts a word are rather awkward to formally state. We will see in Example 8.50 an alternative way of saying whether a word is accepted by a halting deterministic state automaton.

Exercise 4.23 (Solution here). Consider the halting deterministic state automaton shown below:



(4.24)

Let the left state • be 1, the right state • be 2, and the bottom state • be 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 4.21.

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? ◊

For further examples, every graph gives rise to a dynamical system; but to ensure that we are discussing the same concept, let us fix the definition of a graph.

Definition 4.25 (Graph)**.** A *graph* $G \coloneqq (E \rightrightarrows V)$ consists of

- a set E of *edges*;
- a set V of *vertices*;
- a *source function* $s\colon E \to V$ that assigns each edge a source vertex;
- a *target function* $t\colon E \to V$ that assigns each edge a target vertex.

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

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

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

The graph itself induces a dynamical system $\varphi\colon Vy^V \to g$, where $\varphi_1 = \mathrm{id}_V$ and $\varphi^\sharp_v e = t(e)$. So its states as well as its positions are the vertices of the graph, and a direction at a vertex $v \in V$ is an edge $e \in E$ coming out of v that takes us from $v = s(e)$ along the edge e to its target vertex $\varphi^\sharp_v e = t(e)$.

Exercise 4.27 (Solution here). Pick your favorite graph G, and consider the associated dynamical system as in Example 4.26. Draw its labeled transition diagram as in (4.3) or (4.24). ◊

Example 4.28. In Exercise 4.14, we built a file-reader as a Moore machine, where a file is a function $f\colon \mathsf{n} \to \mathsf{ascii}$ from entries to characters. Now we turn that file-reader into a dependent dynamical system $\varphi\colon Sy^S \to p$ with only one direction while reading.

We let $S \coloneqq \{(s,t) \mid 1 \leq s \leq t \leq n\}$, 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:

$$p(1) \coloneqq \{\texttt{ready}, \texttt{busy}\} \times \mathsf{ascii}.$$

So each p-position is a pair (m, c), where $c \in \mathsf{ascii}$ and m is one of two modes: `ready` or `busy`. Then we define the direction-sets of p for each $c \in \mathsf{ascii}$ as follows:

$$p[(\texttt{ready}, c)] \coloneqq S \qquad \text{and} \qquad p[(\texttt{busy}, c)] \coloneqq \{\texttt{advance}\} \cong 1.$$

That way, our file-reader can receive as its direction any pair of entries in S 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 character at the current entry. So we define the return function φ_1 such that, for all $(s, t) \in S$,

$$\varphi_1(s,t) = \begin{cases} (\texttt{ready}, f(s)) & \text{if } s = t \\ (\texttt{busy}, f(s)) & \text{otherwise} \end{cases}$$

While the file-reader is `ready`, we want to set its new current and terminal entries to equal the given direction. So for each $(s, s) \in S$, define the update function $\varphi^\sharp_{(s,s)} \colon S \to S$ 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 $s < t$, we let the update function $\varphi^\sharp_{(s,t)} \colon \mathsf{1} \to S$ specify the element $(s + 1, t) \in S$, thus shifting its current entry up by 1.

Exercise 4.29 (Solution here). Say instead of a file-reader, we wanted a file-searcher, which acts just like the file-reader from Example 4.28 except that it only returns $c \in \mathsf{ascii}$ in its position when c is a specific character; say $c = 100$. Otherwise, it returns the placeholder character _. 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 4.40, we will see a simpler way to construct a file-searcher by leveraging the file-reader we have already defined. Moreover, this construction will highlight precisely how our file-searcher is related to our file-reader. This will be possible using *wrapper interfaces*, which we will introduce in Section 4.3.3.

Example 4.30. Choose $n \in \mathbb{N}$, a *grid size*, and for each $i \in \mathsf{n}$, let D_i be the set

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

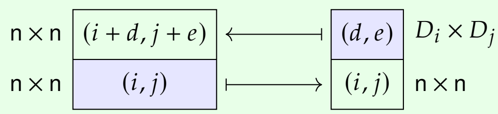
We can think of D_i as the set of ways a robot could move from location i. If $1 < i < n$, a robot may shift its location by -1 (move left/down), 0 (remain still), or $+1$ (move right/up). But a robot already at $i = 1$ cannot shift its location by -1; likewise, a robot already at $i = n$ cannot shift its location by $+1$.

Then we can model a robot told to move in an $\mathsf{n} \times \mathsf{n}$ grid as a dependent dynamical system

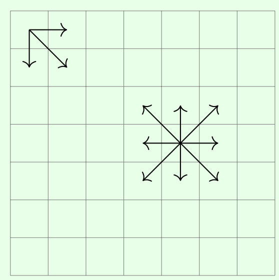
$$\varphi \colon (\mathsf{n}\times\mathsf{n})y^{\mathsf{n}\times\mathsf{n}} \to \sum_{(i,j)\in\mathsf{n}\times\mathsf{n}} y^{D_i\times D_j}.$$

The robot's state is a location $(i, j) \in \mathsf{n}\times\mathsf{n}$ in the grid. We let $\varphi_1 := \mathrm{id}_{\mathsf{n}\times\mathsf{n}}$ so that the dynamical system returns its state as its position: the robot expresses its position by moving to that location in the grid.

For each $(i, j) \in \mathsf{n}\times\mathsf{n}$, we let $\varphi^\sharp_{(i,j)}$ send each pair $(d, e) \in D_i \times D_j$ to the grid location $(i+d, j+e) \in (n, n)$. Concretely, this says that if a robot located at (i, j) receives the pair (d, e) as its direction, its new position will be $(i + d, j + e)$. As polyboxes, the dynamical system is given by



Our definition of D_i for each $i \in \mathsf{n}$ guarantees that this position is still inside the grid; with this setup, the robot has fewer ways to move on the sides or corners of the grid than anywhere else:



In this picture, $\mathsf{n} := 7$, and the 4 directions at position $(1, 7)$ and 9 directions at position $(5, 4)$ are shown (recall that remaining still is an option in either case).

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

Exercise 4.31 (Solution here). Modify the dynamical system from Example 4.30 as follows.

1. Replace the interface with a new polynomial p so that at each grid location, the robot can receive not only the direction it should move in but also a "reward value" $r \in \mathbb{R}$.
2. Replace the state-set with a new set 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 dynamical system $\varphi' \colon Sy^S \to p$ that preserves the behavior of the dynamical system from Example 4.30 while updating the robot's reward list without returning it externally. ◊

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 simpler way to think about the new system: it is the juxtaposition of two systems, a robot system and a reward system, in parallel. We will see how to express this in terms of lenses in Exercise 4.37, once we explain how to juxtapose systems like this in general in Section 4.3.2. In fact, we will see in Exercise 4.38 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 4.30 to more than two dimensions.

4.3 Constructing new dynamical systems from old

We have 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, based only on what we know of **Poly** so far from Chapter 3, 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 4.3.1.
2. Given dynamical systems $\varphi\colon Sy^S \to p$ and $\psi\colon Ty^T \to q$, we can take their parallel product to obtain a dynamical system $\varphi \otimes \psi\colon STy^{ST} \to p \otimes q$; see Section 4.3.2.
3. Given a dynamical system $\varphi\colon Sy^S \to p$ and a lens $f\colon p \to q$, we can compose them to obtain a dynamical system $\varphi \mathbin{⨟} f\colon Sy^S \to q$; see Section 4.3.3.

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

4.3.1 Categorical products: multiple interfaces operating on the same states

Let I be a set, and say that we have an I-indexed family of dependent dynamical systems $(\varphi_i\colon Sy^S \to p_i)_{i\in I}$ that all share the same state-set S. Then since **Poly** has all small products, the universal property of products induces a lens

$$\varphi\colon Sy^S \to \prod_{i\in I} p_i,$$

which is itself a dynamical system with state-set S. By (3.57), the interface of φ (i.e. the product that is its codomain) has position-set

$$\left(\prod_{i\in I} p_i\right)(1) \cong \prod_{i\in I} \big(p_i(1)\big)$$

and, at each position $\bar{j}\colon (i \in I) \to p_i(1)$, direction-set

$$\sum_{i\in I} p_i[\bar{j}i].$$

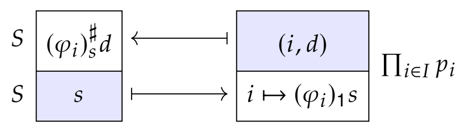
We then characterize the dynamics of φ in terms of each φ_i by leveraging the universal property of products in **Poly**, detailed in the binary case in the solution to Exercise 3.64, as follows. The return function

$$\varphi_1 : S \to \prod_{i \in I} (p_i(1))$$

sends each state $s \in S$ to the dependent function $(i \in I) \to p_i(1)$ sending $i \in I$ to the position $(\varphi_i)_1 s$ returned by the corresponding dynamical system φ_i at the state s. Then the update function

$$\varphi_s^\sharp : \sum_{i \in I} p_i[(\varphi_i)_1 s] \to S$$

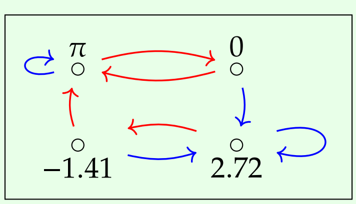
at $s \in S$ sends each pair (i, d) in its domain, with $i \in I$ and $d \in p_i[(\varphi_i)_1 s]$, to where the update function of φ_i at s sends the direction d: namely $(\varphi_i)_s^\sharp d$. We can write φ using polyboxes as follows:

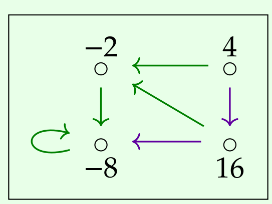


In other words, if there are multiple interfaces that drive the same set of states, we may view them as a single product interface that drives those states. This single dynamical system returns positions in all of the original systems at once; then it can receive a direction from any one of the original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

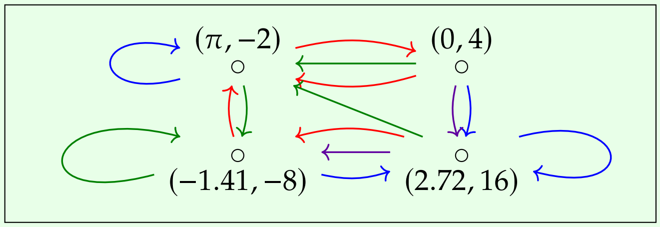
Exercise 4.32 (Solution here). Given a set I, suppose we have an (A_i, B_i)-Moore machine with state-set S for each $i \in I$. Show that there is an induced $(\sum_{i \in I} A_i, \prod_{i \in I} B_i)$-Moore machine, again with state-set S. ◊

Example 4.33. Consider two four-state dependent dynamical systems $\varphi : 4y^4 \to \mathbb{R}y^{\{r,b\}}$ and $\psi : 4y^4 \to \mathbb{Z}_{\geq 0}y^{\{g,p\}} + \mathbb{Z}_{<0}y^{\{g\}}$, 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 positions: one according to the return function of φ, and another according to the return function of ψ. As for the possible directions, we can now choose either a direction of φ (either r or b), in which case the dynamical system will update its state according to the update map of φ; or a direction of ψ (either g or sometimes p), in which case the dynamical system will update its state according to the update map of ψ.

Exercise 4.34 (Toward event-based systems; solution here). Let $\varphi\colon Sy^S \to p$ be a dynamical system. We can think of it as requiring a direction at each time step to update its state.

Suppose we want to change φ into an *event-based system*: one that does not always receive a direction and changes state only when it does. That is, we want every position of p to have an extra direction that, when selected, never changes the state. We want a new system $\varphi'\colon Sy^S \to p'$ that has this behavior; what should p' and φ' be? ◊

4.3.2 Parallel products: juxtaposing dynamical systems

Another way to combine two polynomials—and indeed two lenses—is by taking their parallel product, as in Definition 3.65. 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. The procedure is straightforward: take the product of the state-sets as the new state-set, the product of the position-sets as the new position-set, and the product of the direction-sets at each of position in a tuple of positions as the new direction-set at that tuple.

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

$$\varphi\colon \bigotimes_{i \in \mathsf{n}} \left(S_i y^{S_i}\right) \to \bigotimes_{i \in \mathsf{n}} p_i.$$

By inductively applying Exercise 3.67, we find that the domain of φ is

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

so φ is itself a dependent dynamical system with state-set $\prod_{i \in \mathsf{n}} S_i$. Meanwhile, inductively applying (3.66) yields

$$\left(\bigotimes_{i \in \mathsf{n}} p_i\right)(1) \cong \prod_{i \in \mathsf{n}} \big(p_i(1)\big)$$

as the position-set and, at each position $(j_i)_{i \in \mathsf{n}}$,

$$\prod_{i \in \mathsf{n}} p_i[j_i]$$

as the direction-set of the interface of φ.

We can characterize the dynamics of φ in terms of each constituent dynamical system φ_i as follows. By our proof sketch of Proposition 3.73, the return function

$$\varphi_1 \colon \prod_{i \in \mathsf{n}} S_i \to \prod_{i \in \mathsf{n}} \big(p_i(1)\big)$$

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 positions $((\varphi_i)_1 s_i)_{i \in \mathsf{n}}$ 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

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

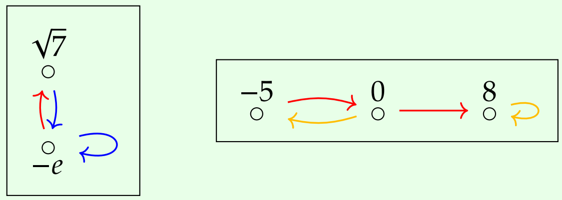
sends each n-tuple of directions $(d_i)_{i \in \mathsf{n}}$ in its domain, with each $d_i \in p_i[(\varphi_i)_1 s_i]$, to the n-tuple $((\varphi_i)^\sharp_{s_i} d_i)_{i \in \mathsf{n}}$ consisting of states to which the update function of each φ_i at s_i sends d_i. We can write φ using polyboxes as follows:

$$\begin{tikzcd}[row sep=small]
\prod_{i \in \mathsf{n}} S_i & \boxed{((\varphi_i)^\sharp_{s_i} d_i)_{i \in \mathsf{n}}} & \boxed{(d_i)_{i \in \mathsf{n}}} \arrow[l, mapsto] & \\
\prod_{i \in \mathsf{n}} S_i & \boxed{(s_i)_{i \in \mathsf{n}}} \arrow[r, mapsto] & \boxed{((\varphi_i)_1 s_i)_{i \in \mathsf{n}}} & \bigotimes_{i \in \mathsf{n}} p_i
\end{tikzcd}$$

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 positions from all of them together; then it can receive directions from all of the constituent systems' direction-sets at those positions 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 4.35 (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 $\prod_{i \in \mathsf{n}} S_i$. ◊

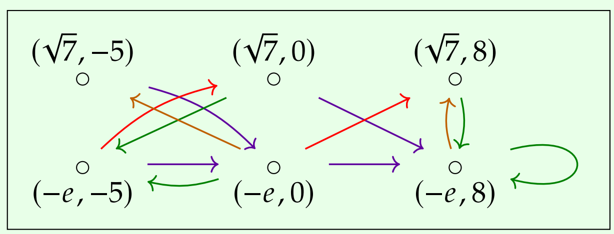
Example 4.36. Consider dependent dynamical systems $\varphi\colon \mathsf{2}y^{\mathsf{2}} \to \mathbb{R}_{<0}y^{\{b,r\}} + \mathbb{R}_{\geq 0}y^{\{b\}}$ and $\psi\colon \mathsf{3}y^{\mathsf{3}} \to \mathbb{Z}_{<0}y^{\{r\}} + \{0\}y^{\{r,y\}} + \mathbb{Z}_{>0}y^{\{y\}}$, drawn below as labeled transition 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

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

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—actually a pair of states from the constituent state-sets—returns two positions, one according to the return function of φ and another according to the return function of ψ. Then every direction must be a pair of directions from the constituent interfaces at those positions, with the update function updating each state in the pair according to each direction in the pair via the constituent update functions of φ and ψ.

Exercise 4.37 (Solution here). Explain how the dynamical system $\varphi'\colon S'y^{S'} \to p'$ you built in Exercise 4.31 can be expressed as the parallel product of the robot-on-a-grid dynamical system $\varphi\colon Sy^S \to p$ from Example 4.30 with another dynamical system, $\psi\colon Ty^T \to q$. Be sure to specify T, q, and ψ. ◊

Exercise 4.38 (Solution here).

1. Explain how the robot-on-a-grid dynamical system $\varphi\colon Sy^S \to p$ from Example 4.30 can be written as the parallel product of some dynamical system with itself.

2. Use k-fold parallel products to generalize Example 4.30 to robots on k-dimensional grids. ◊

Intuitively, the parallel product takes two dynamical systems and puts them in the same room together so that they can be run at the same time. But it does not allow for any interaction *between* the two systems. For that, we will need to use what we call a wrapper interface. We will 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 4.4.

4.3.3 Composing lenses: wrapper interfaces

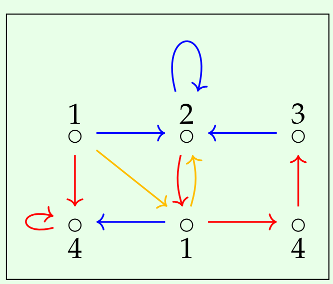
Given a dynamical system $\varphi\colon Sy^S \to p$, say that we want to interact with its state system using a new interface q rather than p. We can do this whenever we have a lens $f\colon p \to q$, which we could compose with our original dynamical system to obtain a new system $Sy^S \xrightarrow{\varphi} p \xrightarrow{f} q$. 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.

How does this new composite system $\varphi \mathbin{⨟} f$ relate to the original dynamical system φ? The lens f converts a position i from p to a position $f_1 i$ from q; at the same time, it allows the choice of direction from $p[i]$ to depend on a choice of direction from $q[f_1 i]$, converting directions of the wrapper interface q to directions of the original interface p. Precomposing f with a dynamical system yields a new dynamical system that lets an agent interact with the original system using only this new interface wrapped around it.

Example 4.39. Consider a dependent dynamical system $\varphi\colon \mathsf{6}y^{\mathsf{6}} \to p$ with

$$p \coloneqq \{1\}y^{\{b,y,r\}} + \{2\}y^{\{b,r\}} + \{3\}y^{\{b\}} + \{4\}y^{\{r\}},$$

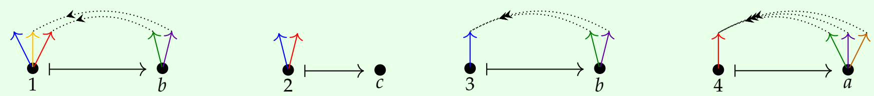
drawn below as a labeled transition diagram (we think of b, y, and r as blue, yellow, and red, respectively):



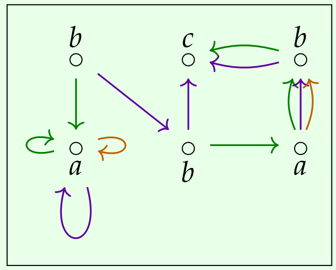
We will wrap the interface

$$q \coloneqq \{a\}y^{\{g,p,o\}} + \{b\}y^{\{g,p\}} + \{c\}$$

around φ using the following lens $f : p \to q$ (we think of g, p, and o as green, purple, and orange, respectively):



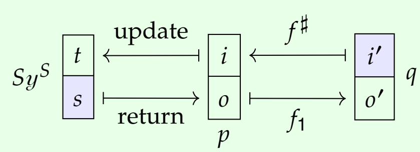
Composing φ with f, we obtain a dynamical system $6y^6 \xrightarrow{\varphi} p \xrightarrow{f} q$ that looks like this:



Each state returns a q-position j according to where the on-positions function of f sends the p-position i that the state returns. Then each $q[j]$-direction is sent to a $p[i]$-direction via the on-directions function of f at i, and the update function of φ uses this $p[i]$-direction to compute the new state. So f allows us to operate φ with the wrapper interface q instead of the original interface p.

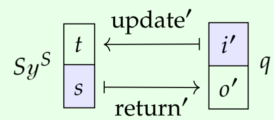
Exercise 4.40 (Solution here). In Exercise 4.29, we built a file-searcher $\psi : Sy^S \to q$ by taking the file-reader $\varphi : Sy^S \to p$ from Example 4.28 and replacing its interface p with a new interface q while keeping its state system Sy^S the same. Express this construction as wrapping q around φ by giving a lens $f : p \to q$ for which composing φ with f yields ψ. ◊

Example 4.41 (Polybox pictures of wrapper interfaces). In polyboxes, composing a dynamical system $\varphi : Sy^S \to p$ with a wrapper $f : p \to q$ looks like this:



The position o displayed by the intermediary interface p is instead exposed as a position $f_1(o) = o'$ of the wrapper interface q in the rightmost position box. Moreover, the direction box of p is no longer blue: an agent who wishes to interact with the middle interface p can only do so via the rightmost interface q. The on-directions function of the wrapper at o converts a direction $i' \in q[o']$ from the rightmost direction box into a direction $i \in p[o]$.

Picture the agent standing to the right of all the polyboxes (i.e. "outside" of the system) with their attention directed leftward (i.e. "inward"), receiving positions from the white position box and feeding directions into the blue direction box. To an agent who is unaware of its inner workings, the composite dynamical system $\varphi \mathbin{;} f$ might as well look like this:



In the next section, we describe a special kind of wrapper.

4.3.4 Sections as wrappers

Say we wanted to model a dynamical system $\varphi\colon Sy^S \to p$ within a closed system, for which an external agent can perceive no change in position and effect no change in direction. We can think of this as wrapping y, the interface with one position and one direction, around φ. To do so, we must specify a wrapper $\gamma\colon p \to y$. In the language of Definition 3.36, this is precisely a *section* of p. As we noted then, this name is appropriate, since γ a way of sectioning off the interface p from the outside world.

Recall that a section $\gamma\colon p \to y$ can be identified with a dependent function of the form $(i \in p(1)) \to p[i]$ that sends each p-position i to a $p[i]$-direction, fixing a direction at every position of p. So a section for an interface dictates the direction it receives given any position it inhabits; there is no need for any further outside interference.

Exercise 4.42 (Solution here). Let $\varphi\colon Sy^S \to By^A$ be an (A, B)-Moore machine.

1. Is it true that a section $\gamma\colon By^A \to y$ can be identified with a function $A \to B$?
2. Describe how to interpret a section $\gamma\colon By^A \to y$ as a wrapper around an interface By^A.
3. Given a section γ, describe the dynamics of the composite Moore machine

$$Sy^S \xrightarrow{\varphi} By^A \xrightarrow{\gamma} y$$

obtained by wrapping y around φ using γ. ◊

Example 4.43 (The do-nothing section). There is something rather off-putting about the way we model dynamical systems as lenses $\varphi\colon Sy^S \to p$. We know that φ sends states-as-positions to positions of the interface and, at each state-as-position, sends directions of the interface to states-as-directions. But we rely only on the labels of elements in S to tell us which positions and directions refer to the same states!

Nothing inherent in the language of **Poly** makes these associations between states-as-positions and states-as-directions for us; we have to rely on the position-set and direction-sets of the state system being the same set for the machine to work properly. Put another way, the monomials $\{4,6\}y^{\{4,6\}}, \{4,6\}y^{\{4,8\}}$, and $\{3,5\}y^{\{6,7\}}$ are all isomorphic in **Poly**, but the first can be a state system while the other two cannot!

To address this issue, we need a way to connect the positions of a polynomial to its own directions in the language of **Poly**. This is where sections can help: a lens $Sy^S \to y$ is just a way of assigning to each position in S a direction in S. So we can define $\epsilon\colon Sy^S \to y$ to be the section that sends each position $s \in S$ to the direction s at s with the same name, corresponding to the same state. Note that ϵ can be identified with the identity function on S (see Exercise 4.42). Now **Poly** knows for each position which direction is associated with the same state the position is.

In this way, we can generalize our notion of state systems to monomials $Sy^{S'}$ equipped with a bijection $S \to S'$, which we can then translate to a section $\epsilon\colon Sy^{S'} \to y$. But for convenience of notation, we will continue to identify the position-set of a state system with each of its direction-sets.

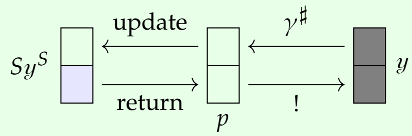
More concretely, the section $\epsilon\colon Sy^S \to y$ acts as a very special (if rather unexciting) dynamical system: it is the *do-nothing section*, with only one possible position and one possible direction that always keeps the current state the same. While the system literally does nothing, we do know one key fact about it: given any state system, regardless of the state-set, we can always define a do-nothing section on it.[a]

Yet this is not the whole story. The do-nothing section knows, at each position, the direction that keeps the system at the same state; but it does not know which of the other directions at that position correspond to which of the other states of the system. We are still relying on the labels of direction-sets being the same for that: for instance, the polynomials $\{1',2',3'\}y^{\{1,2,3\}}$ and $\{1'\}y^{\{0,1,4\}}+\{2'\}y^{\{2,5,6\}}+\{3'\}y^{\{-8,-1,3\}}$ are isomorphic, but even though each has a do-nothing section matching $1' \mapsto 1, 2' \mapsto 2, 3' \mapsto 3$ that makes the first one into a state system, we do not have a way to tell **Poly** how to make the second one a state system yet.

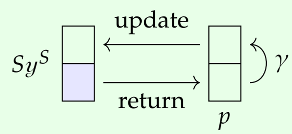
From another perspective, $\epsilon\colon Sy^S \to y$ does nothing, while $\varphi\colon Sy^S \to p$ does "one thing": it steps through the system once, producing the current state's position with the return function and taking in directions with the update function. It is ready to take another step, but how does **Poly** know which state to visit next? Is there a lens that does "two things," "n things," or "arbitrarily many things"? Can we actually *run* a dynamical system in **Poly**? We will develop the machinery to answer these questions over the course of Part II, starting in Section 6.1.4.

[a]It may have bothered you that we call Sy^S, which is a single polynomial, a state *system*, when we also use the word "system" to refer to dependent dynamical systems, which are lenses $Sy^S \to p$. The existence of the do-nothing section explains why our terminology does not clash: every polynomial Sy^S comes equipped with a dependent dynamical system $\epsilon\colon Sy^S \to y$, so it really is a state *system*.

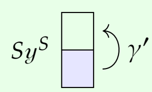
Example 4.44 (Polybox pictures of sections as wrappers). In polyboxes, composing a system $Sy^S \to p$ with a section γ of p can be depicted as



or, equivalently, as



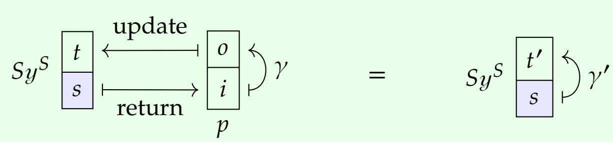
Remember: in a polybox depiction of a dynamical system, the world outside the system exists to the right of all the boxes. So the first picture represents y as a gray wall, cutting off any interaction between the system to its left and the world to its right. Meanwhile, the second picture illustrates how a sectioned-off system independently selects directions of the intermediary interface p via γ according to the p-positions that the inner (leftward) system $Sy^S \to p$ returns. While the second picture shows us why the closed system neither seeks nor requires external directions, the first picture helps remind us that any returned p-positions never reach the outside world either. The composite system is therefore equivalent to the section drawn as follows:



In lens parlance, $\gamma' \colon Sy^S \to y$ is the original system $Sy^S \to p$ composed with $\gamma \colon p \to y$; in the language of dependent functions, $\gamma' \colon S \to S$ is given by

$$\gamma'(s) = \text{update}(s, \gamma(\text{return}(s))) \qquad \text{for all } s \in S,$$

where we interpret γ as a dependent function $(i \in p(1)) \to p[i]$. We can deduce this equation by equating the previous two polybox pictures, knowing they represent the same lens:

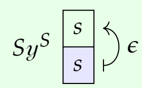


Equating the directions boxes of the domain on either side, we have that $t = t'$, so

$$\gamma'(s) = t' = t = \text{update}(s, o) = \text{update}(s, \gamma(i)) = \text{update}(s, \gamma(\text{return}(s))).$$

Later on we will read more intricate equations off of polyboxes in this manner, although we will not spell out the procedure in so much detail; we encourage you to trace through the arrows on your own.

Example 4.45 (The do-nothing section in polyboxes). In Example 4.43, we saw that every state system Sy^S can be equipped with a section $\epsilon\colon Sy^S \to y$ called the do-nothing section, which assigns each state-as-position to its corresponding state-as-direction, thus leaving the state unchanged. That is, it is the section whose polyboxes can be drawn as follows:



4.4 General interaction

We are now ready to use **Poly** to model interactions between dependent dynamical systems that can change their interfaces and interaction patterns.

4.4.1 Wrapping juxtaposed dynamical systems together

When wrapper interfaces 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 a dynamical system $\varphi_i\colon S_i y^{S_i} \to p_i$ for each $i \in \mathsf{n}$, we can first juxtapose them to form a single dynamical system $\varphi \mathbin{⨾} f$:

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

by taking their parallel product. Then we can wrap an interface q around φ using a wrapper $f\colon \bigotimes_{i \in \mathsf{n}} p_i \to q$, yielding a new composite dynamical system

$$\left(\prod_{i \in \mathsf{n}} S_i\right) y^{\prod_{i \in \mathsf{n}} S_i} \xrightarrow{\varphi} \bigotimes_{i \in \mathsf{n}} p_i \xrightarrow{f} q.$$

On positions, f gives a way of combining all the positions of the constituent interfaces into a single position of the wrapper interface. On directions, f takes into account the current positions of each constituent interface along with a direction for the wrapper interface to specify a direction for each of the constituent interfaces. In particular, a judiciously chosen on-directions function could feed positions of some interfaces as directions to others. When f is a wrapper around a parallel product of interfaces, we call f the *interaction pattern* between those interfaces.

Example 4.46 (Repeater). Suppose we have a dynamical system $\varphi\colon Sy^S \to Ay + y$ that takes unchanging directions and sometimes returns elements of A while other times returning only silence (the position associated with the right hand summand y). 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 was returned. (We allow ψ to repeat an arbitrary element of A if φ returns silence before it has returned any elements of A yet.)

What we need is a way to store an element of A and retrieve it as needed. So whenever φ returns an element of A, 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 directions, return elements of A as positions, 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 its position and updates its state to the direction 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

$$\varphi \otimes \iota\colon Sy^S \otimes Ay^A \cong SAy^{SA} \to (Ay + y) \otimes Ay^A$$

that runs both systems in parallel: simultaneously and independently. But what we want is for φ and ι to interact with each other, and for the resulting system to only return elements of A. To do so, we need to wrap an interface Ay around $\varphi \otimes \iota$ 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 that we must define. Then we will be able to define $\psi \coloneqq (\varphi \otimes \iota) \mathbin{⨟} f$.

Since $\otimes$ distributes over $+$, by the universal property of the coproduct, it suffices to give lenses

$$g\colon Ay \otimes Ay^A \cong (A \times A)y^A \to Ay \qquad \text{and} \qquad h\colon y \otimes Ay^A \cong Ay^A \to Ay.$$

The former corresponds to the case where φ returns an element of A, while the latter corresponds to the case where φ is silent.

When φ returns an element of A, we want the composite system ψ to return that same element, but we also want to give that position as a direction to ι so that it can be stored. We do not need to do anything with the position returned by ι; we can simply discard it. So g should map $(a, a') \mapsto a$ on positions, yielding the position returned by φ and discarding the position returned by ι; and the on-directions function $g^\sharp_{(a,a')}\colon 1 \to A$

should pick out the direction $a \in A$, feeding the position returned by φ as a direction for ι.

On the other hand, when φ returns silence, we want ψ to return the position of ι instead. We also need to feed this position back to ι as a direction so that it can continue to be stored. So h should be the identity on positions as well as the identity on directions.

Example 4.47 (Paddling). Say we wanted to build a Moore machine with interface $\mathbb{N}y$; we will interpret the natural number position it returns as describing the machine's current location. Suppose we want to be very strict about what how far the machine can move and what can make it move.

To model 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 directions) and can only move its paddle (i.e. return a position) `left` or `right`: its position-set is $2 \cong \{\texttt{left}, \texttt{right}\}$. The tracker has interface $\mathbb{N}y^2$ because it will return the location of the machine (as an element $n \in \mathbb{N}$) as its position and take in the position of the paddler (as an element of 2) as its direction. We can wrap an interface $\mathbb{N}y$ around them both using an interaction pattern

$$2y \otimes \mathbb{N}y^2 \cong 2\mathbb{N}y^2 \to \mathbb{N}y$$

whose on-positions function is the canonical projection $2\mathbb{N} \to \mathbb{N}$, returning the location returned by the tracker, and whose on-directions map is the projection $2\mathbb{N} \to 2$, passing the position of the paddler as a direction to the tracker.

Let us leave the paddler's dynamics alone—how you the paddler may behave is arbitrary—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 machine forward one unit. Thus the states of the tracker are given by $T \coloneqq 2\mathbb{N}$, storing what side the paddler is on and the machine's current location. The on-positions function of the tracker is the canonical 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

$$d' \mapsto \begin{cases} (d', i) & \text{if } d = d' \\ (d', i+1) & \text{if } d \neq d', \end{cases}$$

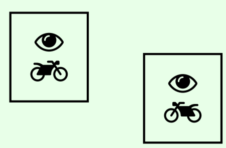
storing the new position of the paddler as well as moving the machine forward one unit if the paddle switches while keeping the machine still if the paddle stays still.

[a]Perhaps one could refer to the tracker as the *demiurge*; it is responsible for maintaining the material universe.

Exercise 4.48 (Solution here). Change the dynamics and state system of the tracker in Example 4.47 so that it exhibits the following behavior.

When the paddle switches once and stops, the tracker increases the location by one unit and stops, as before in Example 4.47. But when the paddle switches twice in a row, the tracker increases the location by two units on the second switch! So if the paddler is stable for a while, then switches three times in a row, the tracker will increase the location by one, then two, then two again. ◊

Example 4.49. Suppose you have two systems with the same interface $p \coloneqq q \coloneqq \mathbb{R}^2 y^{\mathbb{R}^2 - \{(0,0)\}}$.



The ordered pair comprising the position of each interface indicates the location of the corresponding system, while the range of possible directions 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 anything at the same location as the eye itself.

Let us have the two systems approach each other accelerating at a rate equal to the reciprocal of the squared distance between them, modeled with discrete time units. If they finally collide, let us have both systems halt. To do this, we want the wrapper interface to be $\{\texttt{go}\}y + \{\texttt{stop}\}$, so that if the system returns `go`, it can still advance to the next state; but if it returns `stop`, it halts. The wrapper $\mathbb{R}^2 y^{\mathbb{R}^2 - \{(0,0)\}} \otimes \mathbb{R}^2 y^{\mathbb{R}^2 - \{(0,0)\}} \to \{\texttt{go}\}y + \{\texttt{stop}\}$ is given on positions by

$$\big((x_1, y_1), (x_2, y_2)\big) \mapsto \begin{cases} \texttt{stop} & \text{if } x_1 = x_2 \text{ and } y_1 = y_2 \\ \texttt{go} & \text{otherwise.} \end{cases}$$

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 system should have returned the `stop` position and 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 as vectors, 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 vector, and update the current velocity vector by adding an acceleration vector with appropriate magnitude pointing to the other system:

$$\mathbb{R}^2 \times \mathbb{R}^2 \xrightarrow{\text{return}} \mathbb{R}^2$$
$$\big((x, y), (v_x, v_y)\big) \xmapsto{\text{return}} (x, y)$$

$$\mathbb{R}^2 \times \mathbb{R}^2 \times (\mathbb{R}^2 - \{(0,0)\}) \xrightarrow{\text{update}} \mathbb{R}^2 \times \mathbb{R}^2$$
$$\big((x, y), (v_x, v_y), (a, b)\big) \xmapsto{\text{update}} \left(x + v_x, y + v_y, v_x + \frac{a}{(a^2 + b^2)^{3/2}}, v_y + \frac{b}{(a^2 + b^2)^{3/2}}\right)$$

$$\begin{array}{r|c|c|c|l}
\cline{2-2}\cline{4-4}
\mathbb{R}^2 \times \mathbb{R}^2 & \left(x + v_x, y + v_y, v_x + \frac{a}{(a^2+b^2)^{3/2}}, v_y + \frac{b}{(a^2+b^2)^{3/2}}\right) & \xleftarrow{\text{update}} & (a, b) & \mathbb{R}^2 \\
\cline{2-2}\cline{4-4}
\mathbb{R}^2 \times \mathbb{R}^2 & ((x, y), (v_x, v_y)) & \xmapsto[\text{return}]{} & (x, y) & \mathbb{R}^2 - \{(0,0)\} \\
\cline{2-2}\cline{4-4}
\end{array}$$

Exercise 4.50 (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 function called the *distance function* for which $d(x, y) = d(y, x)$, $d(x, y) = 0$ if and only if $x = y$, and $d(x, y) + d(y, z) \geq d(x, z)$ for all $x, y, z \in X$. Let us 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 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. Modify the previous interaction pattern to specify a new interaction pattern $p \otimes p' \to y^{[0,5]}$ where the value $s \in [0, 5]$ is a scalar that changes the distance threshold for the signal to s.
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 $\varphi \colon SXy^{SX} \to AXy^{A'X}$, 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.) $\Diamond$

4.4.2 Sectioning juxtaposed dynamical systems off together

We saw in Section 4.3.4 that a section (i.e. lens to y) for the interface of a dynamical system sections that dynamical system off as a closed system. So it should not come as a surprise that a section 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 4.51 (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 a section for your juxtaposition.

Say that your hand can be at one of two heights, `down` or `up`, and that your fingers can either be `pressed` (with pressure between your thumb and forefinger) or `unpressed`. Say too that you take in information about the chalk's height, which can be `down` or `up` as well. Here are the two sets we will be using:

$$H \coloneqq \{\texttt{down}, \texttt{up}\} \qquad \text{and} \qquad P \coloneqq \{\texttt{pressed}, \texttt{unpressed}\}.$$

Your interface is HPy^H: your position is your own height and pressure, and your possible directions are the chalk's possible heights. As for the chalk, it is either `in` your possession or `out` of it as well as either `down` or `up`. The direction the chalk receives includes whether it is `pressed` or `unpressed`. When it's `out` of your possession, that is the entire direction, but when it is `in` your possession, its direction also comprises your hand's height. In summary, here are the two interfaces:

$$\text{You} \coloneqq HPy^H \qquad \text{and} \qquad \text{Chalk} \coloneqq \{\texttt{out}\}Hy^P + \{\texttt{in}\}Hy^{HP}.$$

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 remains `unpressed`. 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 lens $\gamma\colon \text{You} \otimes \text{Chalk} \to y$, we use the fact that Chalk is a coproduct and that $\otimes$ distributes over coproduct to write You $\otimes$ Chalk as a coproduct itself:

$$\text{You} \otimes \text{Chalk} \cong HPy^H \otimes \{\texttt{out}\}Hy^P + HPy^H \otimes \{\texttt{in}\}Hy^{HP}.$$

Then by the universal property of coproducts, to define γ, it suffices to define two lenses

$$\alpha\colon HPy^H \otimes \{\texttt{out}\}Hy^P \to y \qquad \text{and} \qquad \beta\colon HPy^H \otimes \{\texttt{in}\}Hy^{HP} \to y$$

The lens β, corresponding to when the chalk is `in` your possession, is easy to describe: it can be identified with a function $HPH \to HHP$, and we take it to be the obvious function sending your height and pressure to the chalk and the chalk's height to you; see Exercise 4.53. But α, corresponding to when the chalk is `out` of your possession, is more semantically interesting: it can be identified with a function $HPH \to HP$ given by

$$(h_{\text{You}}, p_{\text{You}}, h_{\text{Chalk}}) \mapsto \begin{cases} (h_{\text{Chalk}}, \texttt{unpressed}) & \text{if } h_{\text{You}} \neq h_{\text{Chalk}} \\ (h_{\text{Chalk}}, p_{\text{You}}) & \text{if } h_{\text{You}} = h_{\text{Chalk}}. \end{cases}$$

In words, this says that if you and the chalk are at different heights, then regardless of your pressure, the chalk remains unpressed; but if you are at the same height as the chalk, the chalk receives your pressure.

Now that you and the chalk are sectioned off together by γ, we are ready to add some dynamics. Your dynamics can be whatever you want, so let us focus on giving dynamics to the chalk (you will be able to give yourself dynamics in Exercise 4.53). The chalk's state is comprised of its height and whether or not it is in your possession, so we give a dynamical system with state-set $C \coloneqq \{\texttt{out}, \texttt{in}\} \times H$ and interface Chalk: that is, a lens

$$\{\texttt{out}, \texttt{in}\} \times Hy^{\{\texttt{out}, \texttt{in}\} \times H} \to \{\texttt{out}\}Hy^P + \{\texttt{in}\}Hy^{HP}. \tag{4.52}$$

On positions, the chalk returns its height and whether it is in your possession directly: in other words, the on-positions function is the identity. On directions, we have two cases. If the chalk is `out` of your possession, it falls `down` unless you catch it, making it `pressed` so that it becomes `in` your possession and retains its current height. So we can express the on-directions function of (4.52) at $(\texttt{out}, h_{\text{Chalk}})$ as

$$\begin{aligned} \texttt{unpressed} &\mapsto (\texttt{out}, \texttt{down}) \\ \texttt{pressed} &\mapsto (\text{'in'}, h_{\text{Chalk}}) \end{aligned}$$

On the other hand, if the chalk is `in` your possession, it takes whatever height it receives from you, remaining `in` your possession if `pressed` but coming `out` of your possession if `unpressed`. So the on-directions function of (4.52) at $(\texttt{in}, h_{\text{Chalk}})$ is given by

$$(h_{\text{You}}, \texttt{unpressed}) \mapsto (\texttt{out}, h_{\text{You}})$$
$$(h_{\text{You}}, \texttt{pressed}) \mapsto (\texttt{in}, h_{\text{You}}).$$

We have thus defined an interaction pattern that allows one system to engage with or disengage from another system and control the behavior of the other system only when the two are engaged.

Exercise 4.53 (Solution here).

1. In Example 4.51, we said that $\beta \colon HPy^H \otimes \{\texttt{in}\}Hy^{HP} \to y$ was easy to describe and given by a function $HPH \to HHP$. Explain what is being said, and provide the function.
2. Provide dynamics to the You interface (i.e. specify a dynamical system with interface You $= HPy^H$) 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 section $p_1 \otimes \cdots \otimes p_n \to y$ sections off these n interfaces together. The following proposition provides an alternative perspective on such sections.

Proposition 4.54. Given polynomials $p, q \in \mathbf{Poly}$, there is a bijection

$$\Gamma(p \otimes q) \cong \mathbf{Set}\big(q(1), \Gamma(p)\big) \ \times \ \mathbf{Set}\big(p(1), \Gamma(q)\big). \tag{4.55}$$

The idea is that specifying a section for the interfaces p and q together is equivalent to specifying a section for p for every output q might return and specifying a section for q for every output p might return.

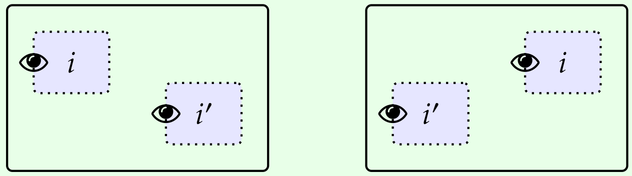
Proof of Proposition 4.54. This is a direct calculation:

$$\begin{aligned}
\Gamma(p \otimes q) &\cong \prod_{i \in p(1)} \prod_{j \in q(1)} (p[i] \times q[j]) \\
&\cong \left(\prod_{j \in q(1)} \prod_{i \in p(1)} p[i] \right) \times \left(\prod_{i \in p(1)} \prod_{j \in q(1)} q[j] \right) \\
&\cong \mathbf{Set}(q(1), \Gamma(p)) \times \mathbf{Set}(p(1), \Gamma(q)).
\end{aligned}$$

□

Example 4.56. A section $f \colon Iy^A \otimes I'y^{A'} \to y$ corresponds to a function $I \times I' \to A \times A'$. In other words, for every pair of positions $(i, i') \in I \times I'$, the section f specifies a pair of directions $(a, a') \in A \times A'$.

Let us think of the positions in I and I' as locations that two machines may occupy.



Then given a pair of locations $(i, i') \in I \times I'$, the interaction pattern f as a function $I \times I' \to A \times A'$ tells us the directions the machines observe, i.e. the ordered pair $(a, a') \in A \times A'$ comprised of what each machine receives. Equivalently, (4.55) says that the interaction pattern tells us what direction the first machine observes at each location when the second machine's location is fixed at i', along with the direction the second machine observes at each location when the first machine's location is fixed at i.

Here we see that (4.55) provides two ways to interpret the interaction pattern between two interfaces in a closed system: either as a section around each interface parametrized by the other's position, or as a single section around them both.

Exercise 4.57 (Solution here). Let $p := q := \mathbb{N}y^{\mathbb{N}}$. We wish to specify a section around their juxtaposition.

1. Say we wanted to feed the position of q as a direction for p. What function $f \colon q(1) \to \Gamma(p)$ captures this behavior?
2. Say we wanted to feed the sum of the positions of p and q as a direction for q. What function $g \colon p(1) \to \Gamma(q)$ captures this behavior?
3. What section $\gamma \colon p \otimes q \to y$ does the pair of functions (f, g) correspond to via (4.55)?
4. Let dynamical systems $\varphi \colon \mathbb{N}y^{\mathbb{N}} \to p$ 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 sectioning φ and ψ off together with γ, i.e. the system $(\varphi \otimes \psi) \mathbin{⨟} \gamma$. ◊

Exercise 4.58 (Solution here). We will use (4.55) to consider the interaction pattern γ between You and Chalk from Example 4.51 as a pair of functions You(1) $\to \Gamma$(Chalk) and Chalk(1) $\to \Gamma$(You).

1. How does the chalk's position specify a section for you? That is, describe the function Chalk(1) $\to \Gamma$(You).
2. How does your position specify a section for the chalk? That is, describe the function You(1) $\to \Gamma$(Chalk). ◊

Exercise 4.59 (Solution here).

1. State and prove a generalization of (4.55) from Proposition 4.54 for n-many polynomials $p_1, \ldots, p_n \in \mathbf{Poly}$.
2. Generalize the "idea" statement between Proposition 4.54 and its proof. ◊

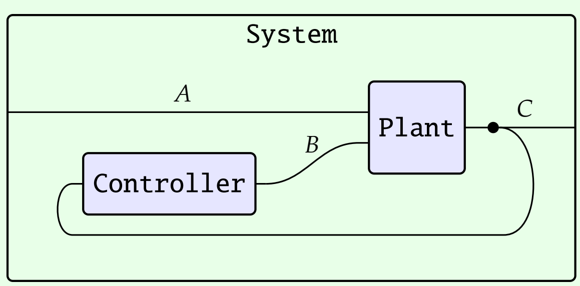
4.4.3 Wiring diagrams as interaction patterns

A *wiring diagram* is a graphical depiction of interactions between systems. Wiring diagrams depict systems as boxes, showing how they send signals 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. The box itself is not a dynamical system, but it becomes a dynamical system once we equip it with a lens from a state system to the interface the box represents. Then the entire wiring diagram—specifying how these boxes nest within a larger box—is just an interaction pattern between the corresponding interfaces, with the larger box playing the role of the wrapper interface. Once every nested box is equipped with a lens from a state system, we obtain a dynamical system whose interface is the larger box.

In the examples to come, we follow the convention that the signals emitted by a box, i.e. positions returned by the corresponding interface, travel along wires out of the right side of that box; while the signals received, i.e. directions observed by the corresponding interface, by a box travel along wires into the left side of that box. A wire may optionally be labeled by the name of the set of elements that may travel as signals along that wire.

Example 4.60. Here is a simple wiring diagram.



(4.61)

The `Plant` is receiving information from the world outside the `System` along the wire labeled A as well as from the `Controller` along the wire labeled B. It is also producing information for the outside world along the wire labeled C which is also being monitored by the `Controller`.

There are three boxes shown in (4.61): the `Controller`, the `Plant`, and the `System`. Each has a fixed set of positions corresponding to the wire(s) connected to its right and a fixed set of directions corresponding to the wire(s) connected to its left, so we can consider each box as a monomial interface, as follows:

$$\texttt{Controller} \coloneqq By^C \qquad \texttt{Plant} \coloneqq Cy^{AB} \qquad \texttt{System} \coloneqq Cy^A. \tag{4.62}$$

Note that in the case of the `Plant`, two wires labeled A and B enter the box from the left, so we take their cartesian product to be the direction-set of the `Plant`.

The wiring diagram itself is a wrapper

$$w\colon \texttt{Controller} \otimes \texttt{Plant} \to \texttt{System},$$

specifying an interaction pattern between the `Controller` and the `Plant` with the `System` as the wrapper interface. Concretely, w is a lens $BCy^{CAB} \to Cy^A$ that prescribes how wires feed positions to directions. As a lens between monomials, w consists of an on-positions function $BC \to C$ and an on-directions map $BCA \to CAB$.

The wiring diagram is a picture that tells us what the on-positions function and on-directions map to use. In particular, the on-positions function sends positions of the inner interfaces to positions of the outer interface, so it is depicted by how the wires coming from the right sides of the inner boxes connect to the right side of the outer box. Given inner boxes that return positions in B and C, the outer box must return a position in C. Here the wire labeled B is not connected to the outer box, but the wire labeled C is, so the on-positions function $BC \to C$ sends $(b, c) \mapsto c$.

Meanwhile, the on-directions function sends positions of the inner interfaces and directions of the outer interface to directions of the inner interfaces, so it is depicted by how the wires coming from both the right sides of the inner boxes and the left side of the outer box connect to the left sides of the inner boxes. Given inner boxes that return positions in B and C and an outer box that receives directions in A, the inner boxes must receive directions in C and B. Again, we can read the on-directions map $BCA \to CAB$ off the wiring diagram: it sends $(b, c, a) \mapsto (c, a, b)$.

Note that neither the wiring diagram nor any of the boxes within it represent 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 we *give dynamics* to the box. So the entire wiring diagram is a wrapper that tells us how, once we give dynamics for each inner box,

$$\varphi\colon Sy^S \to \texttt{Controller} \qquad \text{and} \qquad \psi\colon Ty^T \to \texttt{Plant},$$

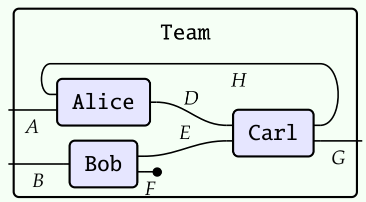
we have given dynamics for the entire outer box:

$$STy^{ST} \xrightarrow{\varphi \otimes \psi} \texttt{Controller} \otimes \texttt{Plant} \xrightarrow{w} \texttt{System}.$$

Exercise 4.63 (Solution here).

1. Make a new wiring diagram like (4.61) except where the controller also receives information from the outside world as an element of a set A'.
2. What are the monomials represented by the boxes in your diagram (replacing (4.62))?
3. What is the interaction pattern represented by this wiring diagram? Give the corresponding lens, including its on-positions and on-directions functions. ◊

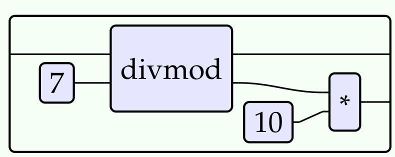
Exercise 4.64 (Solution here). Consider the following wiring diagram.



1. Write out the monomials for `Alice`, `Bob`, and `Carl`.
2. Write out the monomial 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 f?
5. Suppose we have dynamical systems $\alpha\colon Ay^A \to$ `Alice`, $\beta\colon By^B \to$ `Bob`, and $\gamma\colon Cy^C \to$ `Carl`. What is the induced dynamical system with interface `Team`? ◊

Exercise 4.65 (Long division; solution here).

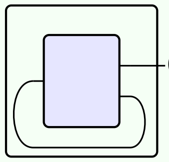
1. Let divmod: $\mathbb{N} \times \mathbb{N}_{\geq 1} \to \mathbb{N} \times \mathbb{N}$ send $(a, b) \mapsto (a \text{ div } b, a \text{ mod } b)$; for example, it sends $(10, 7) \mapsto (1, 3)$ and $(30, 7) \mapsto (4, 2)$. Use Example 4.9 to turn it into a dynamical system.
2. In the following wiring diagram, we have already given dynamics to each box, as follows.



The dynamical system corresponding to the box divmod, with the box as its interface, is the dynamical system from the previous part (the upper wires correspond to the left hand factors of the domain and codomain of the divmod function, while the lower wires correspond to the right hand factors). Similarly, the box labeled $*$ corresponds to the to the dynamical system arising from the multiplication function $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ sending $(m, n) \mapsto mn$ following Example 4.9. Meanwhile the boxes labeled 7 and 10 correspond to dynamical systems with

1 state each that always return $7 \in \mathbb{N}_{\geq 1}$ and $10 \in \mathbb{N}$, respecively. Describe the behavior of the dynamical system corresponding to the entire outer box.

3. Using the outer box from the wiring diagram above as the inner box of the wiring diagram below, pick an initial state so that the resulting dynamical system alternates between returning 0's and the base-10 digits of 1/7 after the decimal point, like so:



$0, 1, 0, 4, 0, 2, 0, 8, 0, 5, 0, 7, 0, 1, 0, 4, 0, 2, 0, 8, 0, 5, 0, 7, 0, 1, 0, 4, 0, 2, 0, 8, 0, 5, 0, 7, \ldots$

We will see in Section 7.1.5 how to make a dynamical system run twice as fast, then apply this to the above system in Example 7.13 so that it skips the 0's. ◊

Example 4.66 (Graphs as wiring diagrams and cellular automata). Suppose we have a graph $G = (E \rightrightarrows V)$ as in Definition 4.25 and a set $\tau(v)$ associated with each vertex $v \in V$:

$$E \underset{t}{\overset{s}{\rightrightarrows}} V \xrightarrow{\tau} \mathbf{Set}$$

We can think of G as an alternative representation of a specific kind of wiring diagram, one in which each inner box has exactly one position wire coming out and the outer box is closed (i.e. no wires in or out, representing the interface y with exactly 1 position and 1 direction). The vertices $v \in V$ are the inner boxes, the set $\tau(v)$ is the position-set associated with the wire coming out of v, and each edge e is a wire connecting the position wire of its target $t(e)$ to a direction wire of its source $s(e)$. An edge from a vertex v_0 to a vertex v_1 indicates that the directions received by v_0 depend on the positions returned by 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 positions and directions, where $E_v \coloneqq s^{-1}(v) \subseteq E$ denotes the set of edges emanating from v. The graph then determines a section

$$\gamma \colon \bigotimes_{v \in V} p_v \to y$$

given by a function

$$\prod_{v \in V} \tau(v) \longrightarrow \prod_{e \in E} \tau(t(e))$$

that sends each dependent function $i \colon (v \in V) \to \tau(v)$ to the dependent function $(e \in E) \to \tau(t(e))$ sending $e \mapsto i(t(e))$. In other words, given the p_v-position $i(v) \in \tau(v)$

returned by each vertex $v \in V$, we know for each edge $e \in E$ that the direction that its source vertex $s(e)$ receives is the position $i(t(e))$ returned by its target vertex $t(e)$.

So once we give dynamics to each p_v, namely by specifying a dynamical system $S_v y^{S_v} \to p_v$ with positions in $\tau(v)$ and directions in $\prod_{e \in E_v} \tau(t(e))$, we will obtain a closed dynamical system that updates the state of each vertex according to the information that they observe from 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*), each with an internal state, in which each vertex $v \in V$ broadcasts a signal (i.e. returns a position) according to its current state, then updates its state according to the signals broadcasted by its *neighbors* in $t(E_v)$ (i.e. positions returned by its neighbors it receives as directions).

For example, many cellular automata have their cells on a 2-dimensional integer lattice. The corresponding graph has vertices $V \coloneqq \mathbb{Z} \times \mathbb{Z}$ and edges given by

$$E \coloneqq (\{-1, 0, 1\} \times \{-1, 0, 1\} - \{(0, 0)\}) \times V,$$

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 swap the roles of the sources and the targets, so that edges point in the direction of data flow rather than in the direction of data dependencies; this is an arbitrary choice.

Exercise 4.67 (Conway's Game of Life; solution here). Conway's Game of Life is a cellular automaton taking place 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 4.66 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 monomials p_v from Example 4.66?
4. What is the appropriate state-set S_v for each interface p_v?
5. What is the appropriate dynamical system lens $S_v y^{S_v} \to p_v$? $\Diamond$

4.4.4 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 static diagram. For example, here we generalize our previous cellular automata example.

Example 4.68 (Generalized cellular automata: voting on who your neighbors are). Recall from Example 4.66 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 target vertices of edges emanating from v. These vertices are the *neighbors* of v, or the vertices that v can "observe." We call the function $N\colon V \to \mathsf{2}^V$ 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 consider only the vertices of our graph and forget the edges. Suppose we are then given a function $n\colon V \to \mathbb{N}$ that we can 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 the monomial each vertex represents is

$$p_v \cong \mathsf{2y}^{\mathsf{2}^{\mathsf{n}(v)}},$$

with its own possible states as positions and its potential neighbors' possible states as directions.

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 4.66, when we had a graph, it told us what the neighbor function should always be. Now we could imagine all the vertices returning their states to vote, via N', on what neighbor function to use to determine which vertices are observing which others.

We can put this all together by providing a section for all the vertices,

$$\bigotimes_{v \in V} p_v \cong \mathsf{2}^V y^{\mathsf{2}^{\sum_{v \in V} n(v)}} \longrightarrow y. \tag{4.69}$$

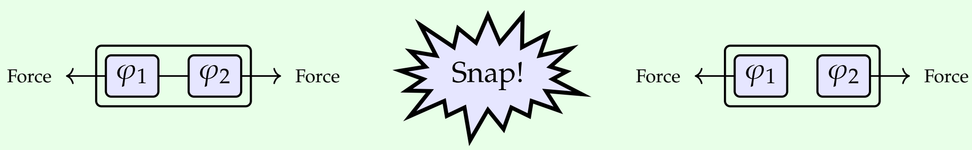
Such a section is equivalent to a function $g\colon \mathsf{2}^V \to \mathsf{2}^{\sum_{v \in V} \mathsf{n}(v)}$ that sends each possible state configuration $S \in \mathsf{2}^V$ of all the vertices in V to a function $g(S)\colon \sum_{v \in V} \mathsf{n}(v) \to \mathsf{2}$ specifying the states every vertex observes. But we already have a neighbor function assigned to S that respects n, namely N'_S: we have $N'_S(v) \cong \mathsf{n}(v)$ 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 $w \in N'_S(v)$. We will let it receive the current state of that neighbor, as given by S:

$$g(S)(v, w) := S(w).$$

We have accomplished our goal: the vertices "vote" on how they should be connected in that their states together determine the neighbor function. We do not mean to imply that this vote needs to be democratic or fair in any way: it is an arbitrary function $N'_{-}: \mathsf{2}^V \to (\mathsf{2}^V)^V$. For instance, the state of a given vertex $v_0 \in V$ may completely determine the neighbor function $V \to \mathsf{2}^V$; this would be expressed by saying that N'_{-} factors as $\mathsf{2}^V \to \mathsf{2}^{\{v_0\}} \cong \mathsf{2} \to (\mathsf{2}^V)^V$, where the left function is the evaluation map at $v_0 \in V$.

Here are some more examples of dependent dynamical systems with interaction patterns in which the way the constituent components are wired together may change.

Example 4.70. In the picture below, forces are applied to the connected boxes on the left; we would like to model how too much force could cause the connection between the boxes to sever, as depicted on the right.



We will imagine the dependent dynamical systems $\varphi_1: Sy^S \to p_1$ and $\varphi_2: Sy^S \to p_2$ 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, their interfaces are given by

$$p_1 := p_2 := Fy^{F \times F} + \{\texttt{snapped}\}y^F,$$

where F is our set of *forces*. We need to be able to add and compare forces, so we need F to be an ordered monoid; let us say $F := \mathbb{N}$ for simplicity. Here each interface has two kinds of positions it can return: either a force $f \in F$ that will be applied to the other interface (i.e. sent to the other interface as a direction) or `snapped`, indicating that the interfaces are no longer connected. The interface always receives a force from the outside world as part of its direction, but when the position of the interface is not `snapped`, it receives a force from the other interface as part of its direction as well. So its direction set is $F \times F$ at positions in F, when the two interfaces are connected; and just F at the position `snapped`, when the two interfaces are not. We define the wrapper interface to be

$$p := y^{F \times F};$$

it can return only 1 position, while its directions are ordered pairs of forces (f_L, f_R) indicating the two external forces acting on the composite system.

Though the systems φ_1 and φ_2 may be initially connected, if the forces on either one surpass a threshold, that system stops communicating with the other system. The connection is broken and neither system ever receives forces from the other again. To implement this explicitly, we need to define an interaction pattern $\kappa\colon p_1 \otimes p_2 \to p$ that wraps p around φ_1 and φ_2. That is, we need to give a lens

$$\kappa\colon (Fy^{F\times F} + \{\mathsf{snapped}\}y^F) \otimes (Fy^{F\times F} + \{\mathsf{snapped}\}y^F) \to y^{F\times F}.$$

Distributing and leveraging coproducts as usual, we find that it suffices to give four lenses:

$$\begin{aligned}
\kappa_{11}\colon \quad & F \times Fy^{(F\times F)(F\times F)} && \to y^{F\times F} \\
\kappa_{12}\colon \quad & F\{\mathsf{snapped}\}y^{(F\times F)F} && \to y^{F\times F} \\
\kappa_{21}\colon \quad & \{\mathsf{snapped}\}Fy^{F(F\times F)} && \to y^{F\times F} \\
\kappa_{22}\colon \quad & \{\mathsf{snapped}\}\{\mathsf{snapped}\}y^{F\times F} && \to y^{F\times F}
\end{aligned} \tag{4.71}$$

The lenses κ_{12} and κ_{21} will not actually occur in our dynamics (when one interface returns $\mathsf{snapped}$, both should), so we take them to be arbitrary. We take the lens κ_{22} to be the obvious isomorphism, passing the external forces to the two internal interfaces. Finally, the lens κ_{11} is equivalent to a function $(F \times F)(F \times F) \to (F \times F)(F \times F)$ which, taking care to remember what each F refers to, we find should send $((f_1, f_2), (f_L, f_R)) \mapsto ((f_L, f_2), (f_1, f_R))$. 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 state-sets for the two systems will be the same, namely $S := F + \{\mathsf{snapped}\}$: each system is either applying a force to the other system or not. The dynamical systems themselves will be the same as well, up to a symmetry swapping left and right; we will define only the left system. It is given by a lens

$$\varphi_1\colon (F + \{\mathsf{snapped}\})y^{F+\{\mathsf{snapped}\}} \to Fy^{F\times F} + \{\mathsf{snapped}\}y^F$$

which we write as the sum of two lenses

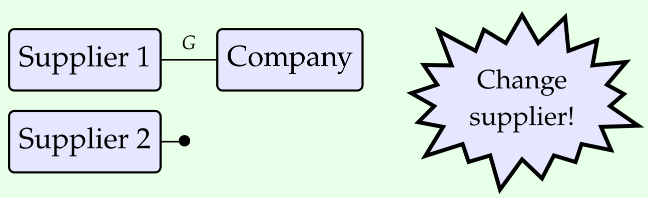
$$Fy^{F+\{\mathsf{snapped}\}} \to Fy^{F\times F} \qquad \text{and} \qquad \{\mathsf{snapped}\}y^{F+\{\mathsf{snapped}\}} \to \{\mathsf{snapped}\}y^F.$$

Both lenses are identities on positions, directly returning their current states. The second lens corresponds to when the connection is broken, after which the connection should remain broken: so its on-directions function is constant, sending any direction to $\mathsf{snapped}$. Meanwhile, the first lens corresponds to the case where the systems are still connected; in this state, the system can receive a pair of forces as its direction and must update its state—either the force it applies or `snapped`—accordingly. We let the on-directions function $F(F \times F) \to F + \{\text{'snapped'}\}$ send

$$(f_1, (f_L, f_2)) \mapsto \begin{cases} \texttt{snapped} & \text{if } f_1 + f_2 \geq 100 \\ f_L & \text{otherwise} \end{cases}$$

Thus, when the sum of forces is above a certain threshold (arbitrarily chosen here to be 100), the internal state is updated to the `snapped` state; otherwise, the internal state is set to the external force applied to the system, which it is now ready to transfer to the other system.

Example 4.72. Consider the case of a company that may change its supplier based on its internal state. The company returns two possible positions, corresponding to whether it wants to receive gizmos in G from the first supplier or widgets in W from the second:



So the company has interface $\{1\}y^G + \{2\}y^W$, the first supplier has interface Gy, and the second supplier has interface Wy. Then a section for the company and the suppliers is a lens

$$(\{1\}y^G + \{2\}y^W) \otimes Gy \otimes Wy \to y,$$

corresponding to a pair of functions $\{1\} \times GW \cong GW \to G$ and $\{2\} \times GW \cong GW \to W$ given by canonical projections. In other words, the company's position determines its supplier and what it receives.

Example 4.73. When someone assembles a machine, their own positions dictate the interaction pattern of the machine's components.

Define $S := \{\texttt{attach}, \texttt{separate}\}$. We say that `unit A` has interface

$$(\{\texttt{attached}\} \times X + \{\texttt{separated}\})\, y^S.$$

It can always receive either the direction to `attach` or to `separate` from S, while its position is either simply `separated` or both `attached` and returning some element of a fixed set X. Meanwhile, `unit B` has interface

$$\{\mathtt{attached}\}y^{X\times S}+\{\mathtt{separated}\}y^{S}.$$

It can also always receive either direction from S, but when it is `attached` it can further receive an element of X. Finally, the role of the person is simply to return whether the units should `attach` or `separate`, so we give it the interface Sy.

Then a section for the person and the units is a lens

$$(\{\mathtt{attached}\}\times X+\{\mathtt{separated}\})\,y^{S}\otimes\left(\{\mathtt{attached}\}y^{X\times S}+\{\mathtt{separated}\}y^{S}\right)\otimes Sy\to y.$$

Such a lens corresponds to four functions, two of which can be arbitrary because our dynamics should never return them (either both units are `attached` or both are `separated`). The other two functions consist of one function

$$\{(\mathtt{attached},\mathtt{attached})\}\times X\times S\cong X\times S\to S\times X\times S$$

that sends $(x,s)\mapsto(s,x,s)$ and another function

$$\{(\mathtt{separated},\mathtt{separated})\}\times S\cong S\to S\times S$$

that sends $s\mapsto(s,s)$.

In words: the person's position tells the units whether they should `attach` or `separate`. If, and only if, the units are `attached`, one unit sends elements of X to the other.

We can easily generalize Example 4.73. 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 can be interpreted as a larger interaction pattern with wrapper interface r:

$$\mathbf{Poly}(p,[q_1\otimes\cdots\otimes q_k\,,\,r])\cong\mathbf{Poly}(p\otimes q_1\otimes\cdots\otimes q_k\,,\,r).$$

In other words, if the positions p returns is deciding the interaction pattern between $q_1,\ldots,q_k$ with wrapper interface r, and the directions p receives is from the subsequent behavior of that interaction pattern itself, then this is equivalent to an interaction pattern with wrapper interface r that p is part of alongside $q_1,\ldots,q_k$.

What it also means is that a dynamical system with interface $[q_1\otimes\cdots\otimes q_k, r]$ is simply selecting interaction patterns $q_1\otimes\cdots\otimes q_k\to r$. Let us see how this works.

4.5 Closure of $\otimes$

The parallel monoidal product is closed—we have a closed monoidal structure on **Poly**—meaning that there is a closure operation, which we denote $[-,-]\colon\mathbf{Poly}^{\mathrm{op}}\times$

Poly → **Poly**, such that there is an isomorphism

$$\mathbf{Poly}(p \otimes q, r) \cong \mathbf{Poly}(p, [q, r]) \tag{4.74}$$

natural in p, q, r. The closure operation is defined on q, r as follows:

$$[q, r] \coloneqq \prod_{j \in q(1)} r \circ (q[j]y) \tag{4.75}$$

Here $\circ$ denotes standard functor composition; informally, $r \circ (q[j]y)$ is the polynomial obtained by replacing 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 cover in greater depth in Part II.

Before we prove that the isomorphism (4.74) holds naturally, let us investigate the properties of the closure operation, starting with some simple examples.

Exercise 4.76 (Solution here). Calculate $[q, r]$ for $q, r \in$ **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 $A \in$ **Set** (constant) and r arbitrary.
5. $q \coloneqq Ay$ for $A \in$ **Set** (linear) and r arbitrary.
6. $q \coloneqq y^2 + 2y$ and $r \coloneqq 2y^3 + 3$. ◊

Exercise 4.77 (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 4.78 (Solution here). Show that there is an isomorphism

$$[q, r] \cong \sum_{f \colon q \to r} y^{\sum_{j \in q(1)} r[f_1 j]} \tag{4.79}$$

where the sum is indexed over $f \in \mathbf{Poly}(q, r)$. ◊

Exercise 4.80 (Solution here). Verify that (3.24) holds. ◊

Example 4.81. 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

$$[p, y] \cong \Gamma(p) y^{p(1)}. \tag{4.82}$$

All these facts follow directly from (4.75).

Exercise 4.83 (Solution here). Verify the three facts above. ◊

Exercise 4.84 (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 4.85. With $[-,-]$ as defined in (4.75), there is a natural isomorphism

$$\mathbf{Poly}(p \otimes q, r) \cong \mathbf{Poly}(p, [q, r]). \tag{4.86}$$

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 4.87 (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 (4.75), or just use (4.86) with the Yoneda lemma and the fact that $y \otimes p \cong p$. ◊

The closure of $\otimes$ implies that for any $q, r \in \mathbf{Poly}$, there is a canonical *evaluation* lens

$$\mathrm{eval}\colon [q,r] \otimes q \to r \tag{4.88}$$

given by sending the identity lens on $[q,r]$ leftward through the natural isomorphism

$$\mathbf{Poly}([q,r] \otimes q, r) \cong \mathbf{Poly}([q,r],[q,r])$$

As in any closed monoidal category, such an evaluation lens has the universal property that for any $p \in \mathbf{Poly}$ and lens $f\colon p \otimes q \to r$, there is a unique lens $f'\colon p \to [q,r]$ such that the following diagram commutes:

$$\begin{tikzcd} p \otimes q \arrow[r, "f' \otimes q"] \arrow[rr, bend right, "f"'] & {[q,r] \otimes q} \arrow[r, "\mathrm{eval}"] & r \end{tikzcd}$$

Exercise 4.89 (Solution here). Obtain the evaluation lens $\mathrm{eval}\colon [p,q] \otimes p \longrightarrow q$ from (4.88). ◊

Exercise 4.90 (Solution here).

1. For any set S, obtain the do-nothing section $Sy^S \to y$ from Example 4.43 whose on-directions is the identity on S using eval and Example 4.81.
2. Show that four lenses in (4.71) from Example 4.70, written equivalently as

$$\begin{aligned} \kappa_{11}\colon\ & Fy^{FF} \otimes Fy^{FF} \to y^F \otimes y^F \\ \kappa_{12}\colon\ & Fy^{FF} \otimes y^F \to y^F \otimes y^F \\ \kappa_{21}\colon\ & y^F \otimes Fy^{FF} \to y^F \otimes y^F \\ \kappa_{22}\colon\ & y^F \otimes y^F \to y^F \otimes y^F, \end{aligned} \tag{4.91}$$

can be obtained by taking the parallel product of identity lenses and evaluation lenses. ◊

Example 4.92 (Modeling your environment without knowing what it is). Imagine a robot whose interface is an arbitrary polynomial q that is part of an interaction pattern $f\colon p \otimes q \to r$ alongside its environment with interface p. Then f induces a lens $f'\colon p \to [q,r]$ such that the following diagram commutes:

$$\begin{tikzcd} p \otimes q \arrow[r, "f' \otimes q"] \arrow[rr, bend right, "f"'] & {[q,r] \otimes q} \arrow[r, "\mathrm{eval}"] & r \end{tikzcd}$$

In other words, $[q, r]$ holds within it all of the possible ways q can interact with other systems when they are wrapped in r together. For example, in the case of $r \coloneqq y$, note that $[q, y] \cong \prod_{i \in q(1)} q[i]y$. That is, a position of $[q, y]$ sends each q-position to a direction at that position, which is exactly what we need to specify to put q in a closed system.

Now suppose we were to give dynamics to q by specifying a lens $Sy^S \to q$. One could aim to choose a set S along with an interesting map $g \colon S \to \mathbf{Poly}(q, r)$. Then each state s would include a guess $g(s)$ about the state of its environment. This is not the real environment p, but just the environment as it affects q, namely $[q, r]$. The robot's states model its environmental conditions.

Example 4.93 (Chu &). Suppose we have polynomials $p_1, p_2, q_1, q_2, r \in \mathbf{Poly}$ and lenses

$$\varphi_1 \colon p_1 \otimes q_1 \to r \qquad \text{and} \qquad \varphi_2 \colon p_2 \otimes q_2 \to r.$$

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 constructed formally via the monoidal closure. We use the closure to rewrite φ_1 and φ_2:

$$\psi_1 \colon p_1 \to [q_1, r] \qquad \text{and} \qquad \psi_2 \colon p_2 \to [q_2, r]$$

Now we take their categorical product to obtain $\psi_1 \times \psi_2 \colon p_1 \times p_2 \to [q_1, r] \times [q_2, r]$. Then we apply Exercise 4.77 to find that $[q_1, r] \times [q_2, r] \cong [q_1 + q_2, r]$, and finally leverage the monoidal closure again to obtain $(p_1 \times p_2) \otimes (q_1 + q_2) \to r$ as desired.

4.6 Summary and further reading

In this chapter we explained how discrete dynamical systems can be expressed as certain lenses between polynomial functors. For example, a Moore machine has an input set A, an output set B, a set of states S, a return function $S \to B$, and an update function $A \times S \to S$. All this is captured in a depedent lens

$$Sy^S \to By^A.$$

We discussed a generalization $Sy^S \to p$, where the output is an arbitrary polynomial $p \in$ **Poly**. We also talked about how to wire machines in parallel by using the parallel product $\otimes$ and how to add wrapper interfaces by composing with lenses $p \to q$.

Throughout the chapter we gave quite a few different examples. For example, we discussed how every function $A \to B$ counts as a memoryless dynamical system. In fact, it was shown in [BPS19] that every dynamical system can be obtained by wiring together memoryless ones. We discussed examples such as file-readers, moving robots, colliding particles, companies that change their suppliers, materials that break when too much force is applied, etc.

For further reading on the mathematics of Moore machines, see [Con12]. For more on mode-dependent interaction, see [ST17]. For a similar and complementary categorical approach to dynamical systems, we recommend David Jaz Myers' *Categorical Systems Theory* book, currently in draft form here: `http://davidjaz.com/Papers/DynamicalBook.pdf`.

4.7 Exercise solutions

Solution to Exercise 4.5.

1. As $S = 3$, the state system is $Sy^S = 3y^3$.
2. As $I = \{0, 1\}$ and $A = \{\text{orange}, \text{green}\}$, the interface is $Iy^A = \{0, 1\}y^{\{\text{orange}, \text{green}\}}$.
3. The return function $S \to I$ sends $L \mapsto 0$, $R \mapsto 1$, and $B \mapsto 1$.
4. The update function $S \times A \to S$ sends

$$\begin{aligned} (L, \text{orange}) &\mapsto L, \quad & (L, \text{green}) &\mapsto R, \\ (R, \text{orange}) &\mapsto L, \quad & (R, \text{green}) &\mapsto B, \\ (B, \text{orange}) &\mapsto L, \quad & (B, \text{green}) &\mapsto B. \end{aligned}$$

5. In the first two steps, the machine sends its bottom state B to its position 1 via its return function, then sends its bottom state B and its direction orange to its left state L via its update function. We can interpret this in terms of the lens φ and depict the steps in polyboxes as

$$\begin{array}{cccccc} S & \boxed{L} & \xleftarrow{\;f^{-1}\;} & \boxed{\text{orange}} & A \\ S & \boxed{B} & \xrightarrow[\;f\;]{} & \boxed{1} & I \end{array}$$

Solution to Exercise 4.8.

At any time, the Moore machine in Example 4.7 is located at a point on the coordinate plane, say $(x, y) \in \mathbb{R}^2$. This location is its current state. When we ask the machine to return its position, it will tell us those coordinates, since the return function is the identity. Then if we give the machine a direction (r, θ) for some distance $r \in [0, 1]$ and angle $\theta \in [0, 2\pi)$, the machine will move by that distance, at that angle counterclockwise from the positive x-axis, from (x, y) to

$$(x + r\cos\theta, y + r\sin\theta) = (x, y) + r(\cos\theta, \sin\theta)$$

(here we treat $\mathbb{R}^2$ as a vector space, so that $r(\cos\theta, \sin\theta)$ is a vector of length r at the angle θ).

Solution to Exercise 4.10.

1. We seek an (A, I)-Moore machine $Iy^I \to Iy^A$ corresponding to the function $f : A \times I \to I$. We know that an (A, I)-Moore machine $Iy^I \to Iy^A$ consists of a return function $I \to I$ and an update

function $I \times A \to I$. So we can simply let the return function be the identity on I and the update function be $I \times A \cong A \times I \xrightarrow{f} B$, i.e. the function f with its inputs swapped. In polyboxes, the machine looks like

$$\begin{array}{rc|c|l}
I & \boxed{f(a,i)} & \longleftarrow\!\!\shortmid & \boxed{a} \; A \\
I & \boxed{i} & \shortmid\!\!\longrightarrow & \boxed{i} \; I
\end{array}$$

2. Generally, such a machine is not memoryless. Unlike in Example 4.9, the update function $I \times A \cong A \times I \xrightarrow{f} I$ does appear to depend on its first input, namely the previous state, which f takes as its second input. We can see this from out polybox picture above: the left direction box, which contains the new state $f(a,i)$, depends on the current state $i \in I$ in the left position box.
However, if f factors through the projection $\pi_1 \colon A \times I \to A$, i.e. if f can be written as a composite $A \times I \xrightarrow{\pi_1} A \xrightarrow{f'} B$ for some $f' \colon A \to B$, then the resulting machine *is* memoryless: it is the memoryless Moore machine from Example 4.9 corresponding to f'.

Solution to Exercise 4.11.

For each of the following constructs, we find $A, I \in \mathbf{Set}$ such that the construct can be identified with a lens $\varphi \colon Sy^S \to Iy^A$, i.e. a return function $\varphi_1 \colon S \to B$ and an update function $\varphi^\sharp \colon S \times A \to S$.

1. Given a discrete dynamical system with states S and transition funtion $n \colon S \to S$, we can set $A \coloneqq I \coloneqq 1$. Then $\varphi_1 \colon S \to 1$ is unique, while $\varphi^\sharp \colon S \times 1 \to S$ is given by $S \times 1 \cong S \xrightarrow{n} S$. The corresponding Moore machine can only be fed one direction (you could think of that direction as a button that simply says "advance to the next state") and can only return one position (which tells us no information). 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 $I \coloneqq S$, so that $\varphi_1 \coloneqq n$ and $\varphi^\sharp \colon S \times 0 \to S$ 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 position, which also 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 $I \coloneqq 1$. Then $\varphi_1 \colon S \to 1$ is unique, while $\varphi^\sharp \colon S \times S \to S$ is equal to m. The corresponding Moore machine always returns the same position. It uses the binary operation m to combine the current state with a given direction—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 direction 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 $I \coloneqq S^S$, so that $\varphi^\sharp \colon S \times 0 \to S$ is unique, while currying m gives φ_1, so that $\varphi_1 s$ is the function $S \to S$ given by $s' \mapsto m(s, s')$. Alternatively, $\varphi_1 s$ could be the function $s' \mapsto m(s', s)$. Either way, this is again 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 $I \coloneqq 2$. Then $\varphi^\sharp \colon S \times 0 \to S$ is unique, while we define $\varphi_1 \colon S \to 2$ by
$$\varphi_1 s = \begin{cases} 1 & \text{if } s \in S' \\ 2 & \text{if } s \notin S' \end{cases}$$
so that S' can be recovered from φ_1 as its fiber over 1. The corresponing Moore machine never moves between its states, but returns one of two positions indicating whether or not the current state is in the subset S'.

Solution to Exercise 4.12.

The original Moore machine had states $\mathbb{R}^2$, so to add a health meter that takes values in $[0,1]$, we take the cartesian product to obtain a new set of states $\mathbb{R}^2 \times [0,1]$. The position-set and direction-set are unchanged, so the Moore machine is a lens

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

Its return function $\mathbb{R}^2 \times [0,1] \to \mathbb{R}^2$ is the canonical projection, as the machine returns only its location in $\mathbb{R}^2$ and not its health; while its update function

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

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. As polyboxes, the lens is

$$\begin{array}{r|c|ccc|c|l}
\cline{2-2}\cline{6-6}
\mathbb{R}^2 \times [0,1] & (x + hr\cos\theta, y + hr\sin\theta, h') & \longleftarrow & & & (r,\theta) & [0,\infty)\times[0,2\pi) \\
\cline{2-2}\cline{6-6}
\mathbb{R}^2 \times [0,1] & (x, y, h) & \longmapsto & & & (x,y) & \mathbb{R}^2 \\
\cline{2-2}\cline{6-6}
\end{array}$$

Solution to Exercise 4.13.

1. The tape has states $S \coloneqq V^{\mathbb{Z}} \times \mathbb{Z}$, positions $I \coloneqq V$, and directions $A \coloneqq V \times \{\texttt{left}, \texttt{right}\}$; as a Moore machine, it is a lens
$$t\colon (V^{\mathbb{Z}} \times \mathbb{Z}) y^{V^{\mathbb{Z}} \times \mathbb{Z}} \to V y^{V \times \{\texttt{left}, \texttt{right}\}}.$$
2. The return function of t should give the value in the current cell of the tape. So $t_1\colon V^{\mathbb{Z}} \times \mathbb{Z} \to V$ is the evaluation map: it sends (f, c) with $f\colon \mathbb{Z} \to V$ and $c \in \mathbb{Z}$ to $f(c) \in V$. Then on a given direction $(v, d) \in V \times \{\texttt{left}, \texttt{right}\}$, the update function of t writes v in the tape's current cell before shifting the current cell number up or down by one according to whether d is `right` or `left`. More precisely,
$$t^\sharp\colon (V^{\mathbb{Z}} \times \mathbb{Z}) \times (V \times \{\texttt{left}, \texttt{right}\}) \to V^{\mathbb{Z}} \times \mathbb{Z}$$
sends current tape $f\colon \mathbb{Z} \to V$, current cell number $c \in \mathbb{Z}$, new value $v \in V$, and $d \in \{\texttt{left}, \texttt{right}\}$ to the new tape $f'\colon \mathbb{Z} \to V$ defined by
$$f'(n) \coloneqq \begin{cases} v & \text{if } n = c \\ f(n) & \text{if } n \neq c \end{cases}$$
and the new cell number $c - 1$ if $d = \texttt{left}$ and $c + 1$ if $d = \texttt{right}$.

Solution to Exercise 4.14.

There are many options for the machine's state-set; we choose to use pairs of entries $(i, t) \in \mathsf{n}^2$, where i is the entry where the file-reader is currently located and t is the entry where the file-reader should stop. We will also include a `stopped` state for when the file-reader has already stopped. So our Moore machine is a lens

$$(n^2 + \{\texttt{stopped}\}) y^{\mathsf{n}^2 + \{\texttt{stopped}\}} \to (\mathcal{ascii} + \{\texttt{done}\}) y^{\{(s,t) \mid 1 \le s \le t \le n\} + \{\texttt{continue}\}}.$$

If the file-reader's current state is `stopped`, then the file-reader should return the position "done." Otherwise, the file-reader should return the character at the entry where the file-reader is currently located. So its return function $\mathsf{n}^2 + \{\texttt{stopped}\} \to \mathsf{ascii} + \{\texttt{done}\}$ sends (i, t) to $f(i)$ and `stopped` to `done`. Meanwhile, the update function

$$(\mathsf{n}^2 + \{\texttt{stopped}\}) \times (\{(s,t) \mid 1 \le s \le t \le n\} + \{\texttt{continue}\}) \to \mathsf{n}^2 + \{\texttt{stopped}\}$$

behaves as follows on the current state and given direction: regardless of the current state, if the given direction is a pair (s, t), the new state will also be (s, t). On the other hand, if the given direction is `continue` and the current state is a pair (i, t), the new state should be the pair $(i+1, t)$ if $i + 1 \leq t$ and `done` otherwise. Finally, if the given direction is `continue` and the current state is `stopped`, the new state should still be `stopped`.

Solution to Exercise 4.23.

1. The set of states is $S \coloneqq \{1, 2, 3\} = \mathsf{3}$ (or equivalently $S \coloneqq \{\bullet, \bullet, \bullet\} \cong \mathsf{3}$).
2. The set of input symbols is $A \coloneqq \{\text{orange}, \text{green}\}$.
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 $\varphi\colon Sy^S \to y^A + 1$, or $\varphi\colon \mathsf{3}y^\mathsf{3} \to y^{\{\text{orange}, \text{green}\}} + 1$. According to the previous part, φ has a return function $\varphi_1\colon S \to \mathsf{2}$ sending states 1 and 2, as non-accept states, to 1; and sending state 3, as an accept state, to 2. Then $\varphi_3^\sharp$ is vacuous, while the other two update functions are given by the the targets of the arrows in (4.24) as follows:
$$\varphi_1^\sharp(\text{orange}) \coloneqq 2, \varphi_1^\sharp(\text{green}) \coloneqq 1;$$
$$\varphi_2^\sharp(\text{orange}) \coloneqq 3, \varphi_2^\sharp(\text{green}) \coloneqq 1,$$
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 4.27.

No matter what graph you chose, Example 4.26 tells us that if you were to draw 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 4.29.

We give a file-searcher $\psi\colon Sy^S \to q$ according to the specification as follows. Its possible positions should form the set
$$q(\mathsf{1}) \coloneqq \{\texttt{ready}, \texttt{busy}\} \times \{100, \_\}.$$
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
$$q[(\texttt{ready}, c)] \coloneqq S \qquad \text{and} \qquad q[(\texttt{busy}, c)] \coloneqq \mathsf{1}.$$
Then we set the return function ψ_1 to behave like φ_1, but with characters not equal to 100 replaced with _: so for all $(s, t) \in S$,
$$\psi_1(s, t) = \begin{cases} (\texttt{ready}, 100) & \text{if } s = t \text{ and } f(s) = 100 \\ (\texttt{ready}, \_) & \text{if } s = t \text{ and } f(s) \neq 100 \\ (\texttt{busy}, 100) & \text{if } s \neq t \text{ and } f(s) = 100 \\ (\texttt{busy}, \_) & \text{otherwise} \end{cases}$$
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 $\psi^\sharp_{(s,t)}\colon \mathsf{1} \to S$ specify the element $(s + 1, t) \in S$, thus shifting its current entry up by 1.

Solution to Exercise 4.31.

We modify the dynamical system from Example 4.30.

1. Previously, the direction-set at position $(i, j) \in \mathsf{n} \times \mathsf{n}$ of our interface was $D_i \times D_j$. But now we also want to give the robot a "reward value" $r \in \mathbb{R}$. So our new direction-set should be $D_i \times D_j \times \mathbb{R}$:
$$p := \sum_{(i,j) \in \mathsf{n} \times \mathsf{n}} y^{D_i \times D_j \times \mathbb{R}}.$$
2. Previously, a state was just a location in the grid: an element of $\mathsf{n} \times \mathsf{n}$. But now we want to be able to record a list of reward values as well. Since each reward value is a real number, we define the state-set to be $S := \mathsf{n} \times \mathsf{n} \times \mathsf{List}(\mathbb{R})$.
3. The former return function φ_1 was the identity on $\mathsf{n} \times \mathsf{n}$. The new return function φ'_1 should still just return 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 $\varphi'_1 \colon \mathsf{n} \times \mathsf{n} \times \mathsf{List}(\mathbb{R}) \to \mathsf{n} \times \mathsf{n}$.

   For each former state $(i, j) \in \mathsf{n} \times \mathsf{n}$, the former update function $\varphi^\sharp_{(i,j)} \colon D_i \times D_j \to \mathsf{n} \times \mathsf{n}$ 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 $(\varphi')^\sharp_{(i,j,(r_1,\ldots,r_k))} \colon D_i \times D_j \times \mathbb{R} \to \mathsf{n} \times \mathsf{n} \times \mathsf{List}(\mathbb{R})$ sends $(d, e, r) \mapsto (i + d, j + e, (r_1, \ldots, r_k, r))$, also updating the list of rewards. As polyboxes, the new dynamical system is given by

$$\begin{array}{r|c|ccc|c|l}
\mathsf{n} \times \mathsf{n} \times \mathsf{List}(\mathbb{R}) & (i + d, j + e, (r_1, \ldots, r_k, r)) & \longleftarrow\!\!\!\shortmid & (d, e, r) & D_i \times D_j \times \mathbb{R} \\
\mathsf{n} \times \mathsf{n} \times \mathsf{List}(\mathbb{R}) & (i, j, (r_1, \ldots, r_k)) & \longmapsto & (i, j) & \mathsf{n} \times \mathsf{n}
\end{array}$$

Solution to Exercise 4.32.

We are given a lens $Sy^S \to B_i y^{A_i}$ for each $i \in I$. The universal property of products in **Poly** then induces a lens
$$Sy^S \to \prod_{i \in I} B_i y^{A_i}.$$
By (3.57), its codomain is the product of monomials
$$\prod_{i \in I} B_i y^{A_i} \cong \left(\prod_{i \in I} B_i \right) y^{\sum_{i \in I} A_i},$$
which, in particular, is still a monomial. Hence the induced lens is a $(\sum_{i \in I} A_i, \prod_{i \in I} B_i)$-Moore machine with state-set S.

Solution to Exercise 4.34.

Given a dynamical system $\varphi \colon Sy^S \to p$, we seek a new dynamical system $\varphi' \colon Sy^S \to p'$ with an additional direction that does not change the state. 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 direction that does not change the state. This latter interface also need not distinguish between its positions; it should have a single position that provides no additional information. So the second interface we want acting on Sy^S is y.

If y were the only interface acting on the state 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 should be the identity function on S, since the direction never changes the state. Then p' is the product of the two interfaces p and y, while $\varphi' \colon Sy^S \to p'$ is the unique lens induced by $\varphi \colon Sy^S \to p$ and $\epsilon \colon Sy^S \to y$. In particular, $p' \cong py \cong \sum_{i \in p(1)} y^{p[i]+1}$, and if we let $*$ denote the additional direction not in p at each position of py, we can write φ' in polyboxes as

$$\begin{array}{r|c|c|c|l}
S & \varphi^\sharp_s a & \longleftarrow\!\!\!\shortmid & a & \\
S & s & \longmapsto & \varphi_1 s & py
\end{array}$$

when the chosen direction is from the original p (i.e. $a \in p[\varphi_1 s]$ above), coinciding with the behavior of the original system φ; or as

$$\begin{array}{cc|c|ccc|c|c} S & & \boxed{s} & \leftarrow\!\!\!\longmapsfrom & & \boxed{*} & & \\ S & & \boxed{s} & \longmapsto & & \boxed{\varphi_1 s} & & py \end{array}$$

when the chosen direction is the additional direction $*$ that does not change the state. It turns out this construction is universal; it is known as *copointing*.

Solution to Exercise 4.35.

We are given a lens $S_i y^{S_i} \to B_i y^{A_i}$ for each $i \in \mathsf{n}$. By inductively applying Exercise 3.67, we find that their parallel product in **Poly** is 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 $(\prod_{i \in \mathsf{n}} A_i, \prod_{i \in \mathsf{n}} B_i)$-Moore machine with state-set $\prod_{i \in \mathsf{n}} S_i$. This works because the parallel product of monomials is still a monomial.

Solution to Exercise 4.37.

We will show that taking the parallel product of the robot-on-a-grid dynamical system $\varphi\colon Sy^S \to p$ from Example 4.30 and a reward-tracking dynamical system $\psi\colon Ty^T \to q$ we will define shortly yields the dynamical system $\varphi'\colon S'y^{S'} \to p'$ from Exercise 4.31.

The reward-tracking dynamical system should have states in $\mathsf{List}(\mathbb{R})$ to record a list of reward values, unchanging position, and directions 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 map sends each state $(r_1, \ldots, r_k) \in \mathsf{List}(\mathbb{R})$ and each direction $r \in \mathbb{R}$ to the new state $(r_1, \ldots, r_k, r)$.

Then the dynamical system from Exercise 4.31 is the parallel product of the robot-on-a-grid dynamical system from Example 4.30 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 4.31.

Solution to Exercise 4.38.

1. The robot-on-a-grid dynamical system from Example 4.30 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 $\lambda_1 \coloneqq \mathrm{id}_{\mathsf{n}}$ 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 4.40.

We give a lens $f\colon p \to q$ for which composing the file-reader $\varphi\colon Sy^S \to p$ from Example 4.28 with f yields the file-searcher $\psi\colon Sy^S \to q$ from Exercise 4.29. The file-searcher returns the same position as the file-reader when the second coordinate of that position 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

$$f_1(m, c) \coloneqq \begin{cases} (m, c) & \text{if } c = 100 \\ (m, \_) & \text{otherwise.} \end{cases}$$

Then the file-searcher acts just like the file-reader does on inputs, so every on-directions function of f should be an identity function.

Solution to Exercise 4.42.

1. No, it represents a function $B \to A$! A section sends each position $b \in B$ to a direction $a \in A$.
2. As a wrapper around an interface By^A, a section $\gamma\colon By^A \to y$ corresponds to a function $g\colon B \to A$ that feeds the direction $g(b) \in A$ into the system whenever it returns the position $b \in B$.
3. Composing our original Moore machine $Sy^S \to By^A$ with a section γ yields a Moore machine $Sy^S \xrightarrow{\varphi} By^A \xrightarrow{\gamma} y$ 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 section γ with a function $g\colon B \to A$, then the composite Moore machine $Sy^S \xrightarrow{\varphi} By^A \xrightarrow{\gamma} y$ 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 position the original state returns, the direction that the section gives in response to that position, and the update function that sends the original state and the selected direction to the new state.

Solution to Exercise 4.48.

We define a new tracker $T'y^{T'} \to \mathbb{N}y^2$ based on the one from Example 4.47 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 $T \coloneqq 2 \times 2 \times \mathbb{N}$, 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 canonical projection $2 \times 2 \times \mathbb{N} \to \mathbb{N}$ that returns the current location; then at each $(d, d', i) \in 2 \times 2 \times \mathbb{N}$, the on-directions function of the tracker $2 \to 2 \times 2 \times \mathbb{N}$ 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 4.50.

1. An interaction pattern $p \otimes p' = AXy^{A'X} \otimes A'Xy^{AX} \to y$ consists of a trivial on-positions function $AX \times A'X \to 1$ and an on-directions map $AX \times A'X \to A'X \times AX$ indicating what directions the robots should receive according to the positions they return. To model the robots receiving each others' locations but only receiving each others' signals when the distance between their locations is less than 1, 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 changes the distance threshold for the signal to $s \in [0, 5]$ consists of a still trivial on-positions function and an on-directions map $AX \times A'X \times [0, 5] \to A'X \times AX$ indicating what directions the robots should receive according to the external direction $s \in [0, 5]$ and the positions they return. To model the fact that the robots receive each others'

locations, but only receive each others' signals when the distance between their locations is less than s, 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 $\varphi\colon SXy^{SX} \to AXy^{A'X}$ 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

$$\varphi\colon DSXy^{DSX} \to \{\text{'close'}\}AXy^{DA'X} + \{\text{'far'}\}AXy^{DX}$$

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 4.53.

1. A lens $HPy^{H} \otimes \{\texttt{in}\}Hy^{HP} \to y$ is a section of $HPy^{H} \otimes \{\texttt{in}\}Hy^{HP} \cong HPHy^{HHP}$ and can thus be identified with a function from its position-set to its direction-set: $HPH \to HHP$. Here the first factor of the domain refers to your height, the second factor to the your pressure, and the third factor to the chalk's height; while in the codomain, the first factor refers to the chalk's height that you receive, the second factor refers to your height that the chalk receives, and the third factor refers to your pressure that the chalk receives. We should therefore define the function $HPH \to HHP$ to be the isomorphism that sends $(h_{\text{You}}, p_{\text{You}}, h_{\text{Chalk}}) \mapsto (h_{\text{Chalk}}, h_{\text{You}}, p_{\text{You}})$.

2. If you alwayss cycle through three possible actions—reaching `down` and grabbing the chalk so that it is `pressed`, moving your hand `up` while keeping it `pressed` to lift the chalk, and dropping the chalk by leaving it `unpressed` while your hand is still `up`—you only need 3 possible states. So we can provide your dynamics using a lens $\mathsf{3}y^{\mathsf{3}} \to \text{You} = HPy^H$. The return function $\mathsf{3} \to HP$ indicates what happens at each state, sending $1 \mapsto (\texttt{down}, \texttt{pressed}), 2 \mapsto (\texttt{up}, \texttt{pressed})$, and $3 \mapsto (\texttt{up}, \texttt{unpressed})$. Meanwhile the update function $\mathsf{3}H \to \mathsf{3}$ always changes the state to the next one, regardless of direction: it ignores the H coordinate and sends $1 \mapsto 2, 2 \mapsto 3$, and $3 \mapsto 1$.

Solution to Exercise 4.57.

1. The function $f\colon q(1) \to \Gamma(p)$ should send each q-position b to the section of p corresponding to the constant function $\mathbb{N} \to \mathbb{N}$ that sends every p-position to b itself. That is, f is the function $b \mapsto (\_ \mapsto b)$.
2. The function $g\colon p(1) \to \Gamma(q)$ should send each position p-position a to the section of q corresponding to the function $\mathbb{N} \to \mathbb{N}$ that sends each q-position b 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 $(a, b) \mapsto ((fb)a, (ga)b) = (b, a + b)$. Then $\gamma\colon p \otimes q \to y$ is the section with this function as its on-directions function.
4. As φ and ψ are both the identity, their parallel product is the identity as well, so $(\varphi \otimes \psi) \mathbin{⨟} \gamma = \gamma$. From the previous part, if the current state of this system is (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 4.58.

1. Fix a position $(s_{\text{Chalk}}, h_{\text{chalk}}) \in \text{Chalk}(1) = \{\texttt{out}, \texttt{in}\}H$ of the chalk. If $s_{\text{Chalk}} = \texttt{out}$, then the corresponding section $\text{You} \cong HPy^H \to y$ given by f via (4.55) can be thought of as the function $HP \to H$ sending
$$(h_{\text{You}}, p_{\text{You}}) \mapsto h_{\text{Chalk}},$$
according to the behavior of $\alpha\colon HPy^H \otimes Hy^P \to y$ when we fix the position of the domain's right factor to be $h_{\text{chalk}} \in H$ and focus on the direction in H of the domain's left factor. Meanwhile, if $s_{\text{Chalk}} = \texttt{in}$, then the corresponding section $\text{You} \cong HPy^H \to y$ can also be thought of as the function $HP \to H$ sending
$$(h_{\text{You}}, p_{\text{You}}) \mapsto h_{\text{Chalk}},$$
according to the behavior of $\beta\colon HPy^H \otimes Hy^{HP} \to y$ when we again fix the position of the domain's right factor to be $h_{\text{chalk}} \in H$ and focus on the direction in H of the domain's left factor. So overall, the desired function $\text{Chalk}(1) \to \Gamma(\text{You})$ is given by
$$(\_, h_{\text{chalk}}) \mapsto ((\_, \_) \mapsto h_{\text{Chalk}}).$$
2. Fix a position $(h_{\text{You}}, p_{\text{You}}) \in \text{You}(1) = HP$ of the chalk. Then the corresponding section $\text{Chalk} \cong \{\texttt{out}\}Hy^P + \{\texttt{in}\}Hy^{HP} \to y$ can be thought of as a pair of functions: one $\{\texttt{out}\}H \to P$ sending
$$(\texttt{out}, h_{\text{Chalk}}) \mapsto \begin{cases} \texttt{unpressed} & \text{if } h_{\text{You}} \neq h_{\text{Chalk}} \\ p_{\text{You}} & \text{if } h_{\text{You}} = h_{\text{Chalk}}, \end{cases}$$
according to the behavior of $\alpha\colon HPy^H \otimes Hy^P \to y$ when we fix the position of the domain's left factor to be $(h_{\text{You}}, p_{\text{You}}) \in HP$ and focus on the direction in P of the domain's right factor; and another $\{\texttt{in}\}H \to HP$ sending
$$(\texttt{in}, h_{\text{Chalk}}) \mapsto (h_{\text{You}}, p_{\text{You}})$$
according to the behavior of $\beta\colon HPy^H \otimes Hy^{HP} \to y$ when we again fix the position of the domain's left factor to be $(h_{\text{You}}, p_{\text{You}}) \in HP$ and focus on the direction in HP of the domain's

right factor. So overall, the desired function $\mathrm{You}(1) \to \Gamma(\mathrm{Chalk})$ is given by

$$(h_{\mathrm{You}}, p_{\mathrm{You}}) \mapsto \left(\begin{array}{l} (\mathtt{out}, h_{\mathrm{Chalk}}) \mapsto \begin{cases} \mathtt{unpressed} & \text{if } h_{\mathrm{You}} \neq h_{\mathrm{Chalk}} \\ p_{\mathrm{You}} & \text{if } h_{\mathrm{You}} = h_{\mathrm{Chalk}} \end{cases} \\ (\mathtt{in}, h_{\mathrm{Chalk}}) \mapsto (h_{\mathrm{You}}, p_{\mathrm{You}}) \end{array} \right).$$

Solution to Exercise 4.59.

1. We generalize (4.55) 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 tautological, and the $n = 2$ case is given by (4.55). Then by induction on n, we have

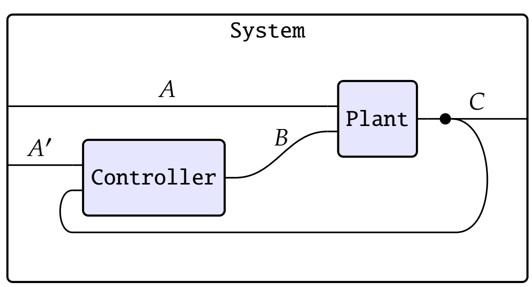
$$\begin{aligned} \Gamma\left(\bigotimes_{i=1}^{n} p_i\right) &\cong \mathbf{Set}\left(p_n(1), \Gamma\left(\bigotimes_{i=1}^{n-1} p_i\right)\right) \times \mathbf{Set}\left(\prod_{i=1}^{n-1} \left(p_i(1)\right), \Gamma(p_n)\right) && (4.55) \\ &\cong \mathbf{Set}\left(p_n(1), \prod_{i=1}^{n-1} \mathbf{Set}\left(\prod_{\substack{1 \leq j \leq n-1, \\ j \neq i}} p_j(1), \Gamma(p_i)\right)\right) \times \mathbf{Set}\left(\prod_{i=1}^{n-1} p_i(1), \Gamma(p_n)\right) \\ & && \text{(Inductive hypothesis)} \\ &\cong \prod_{i=1}^{n-1}\left(\mathbf{Set}\left(\prod_{\substack{1 \leq j \leq n, \\ j \neq i}} p_j(1), \Gamma(p_i)\right)\right) \times \mathbf{Set}\left(\prod_{i=1}^{n-1} p_i(1), \Gamma(p_n)\right), \\ & && \text{(Universal properties of products and internal homs)} \end{aligned}$$

and the result follows.

2. The general idea is that specifying a section for interfaces $p_1, \ldots, p_n$ together is equivalent to specifying a section for p_i for every combination of positions that all the other interfaces might return together, for each $i \in \mathsf{n}$.

Solution to Exercise 4.63.

1. Here is the wiring diagram (4.61) modified so that the controller also receives information from the outside world as an element of A'.



2. The monomials represented by the boxes in this diagram are the same, except that the `Controller` and the `System` both have extra A' factors in their exponent:

$$\mathtt{Controller} := By^{A'C} \qquad \mathtt{Plant} := Cy^{AB} \qquad \mathtt{System} := Cy^{AA'}.$$

3. The interaction pattern represented by this wiring diagram is the lens

$$w' \colon \texttt{Controller} \otimes \texttt{Plant} \to \texttt{System}$$

consisting of an on-positions function $BC \to C$ given by $(b, c) \mapsto c$ and an on-directions function $BCAA' \to A'CAB$ given by $(b, c, a, a') \mapsto (a', c, a, b)$.

Solution to Exercise 4.64.

1. According to the wiring diagram, we have that $\texttt{Alice} \coloneqq Dy^{HA}$, that $\texttt{Bob} \coloneqq EFy^{B}$, and that $\texttt{Carl} \coloneqq HGy^{DE}$.
2. According to the wiring diagram, we have that $\texttt{Team} \coloneqq Gy^{AB}$.
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 4.65.

1. Using Example 4.9, we can turn divmod into the dynamical system $\text{divmod} \colon \mathbb{N} \times \mathbb{N}y^{\mathbb{N}\times\mathbb{N}} \to \mathbb{N} \times \mathbb{N}y^{\mathbb{N}\times\mathbb{N}_{\geq 1}}$ whose return function is the identity on $\mathbb{N} \times \mathbb{N}$ and whose update map $\mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}_{\geq 1} \to \mathbb{N} \times \mathbb{N}$ 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 position 7; similarly, the box labeled 10 is given dynamics $10 \colon y \to \mathbb{N}y$ so that it always returns the position 10. Meanwhile, the box labeled divmod is given dynamics $\text{divmod} \colon \mathbb{N} \times \mathbb{N}y^{\mathbb{N}\times\mathbb{N}} \to \mathbb{N} \times \mathbb{N}y^{\mathbb{N}\times\mathbb{N}_{\geq 1}}$ from the previous part; and applying Exercise 4.10 to the multiplication function $* \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ yields the dynamics for the box labeled $*$: 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 map $\mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ sends $(\_, m, n) \mapsto mn$.
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 map $(s, q, r, t, p, a) \mapsto (a, s, r, t)$. 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 $a \mapsto (a, 7, r, 10) \mapsto (a \text{ div } 7, a \text{ mod } 7, 10r)$.
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 of the second wiring diagram is the composite lens

$$\left(\mathbb{N} \times \mathbb{N}y^{\mathbb{N}\times\mathbb{N}}\right) \otimes \mathbb{N}y^{\mathbb{N}} \xrightarrow{\varphi} \mathbb{N} \times \mathbb{N}y^{\mathbb{N}} \xrightarrow{w'} \mathbb{N}y$$

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) specifies the new state $(p \text{ div } 7, p \bmod 7, 10r)$.

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 we set the initial state to be $(q, r, p) \coloneqq (0, 0, 10)$, then the subsequent states will be as follows, with the values of q in the left column giving us the positions returned:

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
$\vdots$	$\vdots$	$\vdots$

Solution to Exercise 4.67.

1. Following the suggestion from the end of Example 4.66, we can use a graph with $V \coloneqq \mathbb{Z} \times \mathbb{Z}$ and $E \coloneqq (\{-1, 0, 1\} \times \{-1, 0, 1\} - \{(0, 0)\}) \times V$ 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 observes only its eight immediate neighbors.
2. Each vertex only needs to return whether it is live or dead, so we assign $\tau(v) \coloneqq \{\texttt{live}, \texttt{dead}\}$ for every $v \in V$.
3. For each $v \in V$, the monomial represented by v from Example 4.66 can be written as
$$p_v \cong \{\texttt{live}, \texttt{dead}\}y^{\mathbf{Set}(\{-1,0,1\}\times\{-1,0,1\}-\{(0,0)\},\, \{\texttt{live}, \texttt{dead}\})}.$$
Every vertex returns either `live` or `dead` as its position and receives as its direction whether each of its eight neighbors is `live` or `dead`.
4. Each vertex $v \in V$ only needs to record whether it is `live` or `dead`, so $S_v \coloneqq \{\texttt{live}, \texttt{dead}\}$.
5. The appropriate dynamical system lens $S_v y^{S_v} \to p_v$ for each vertex $v \in V$ should have the identity function on $\{\texttt{live}, \texttt{dead}\}$ as its return function, while its update map should be a function
$$\{\texttt{live}, \texttt{dead}\} \times \mathbf{Set}(\{-1, 0, 1\} \times \{-1, 0, 1\} - \{(0, 0)\}, \{\texttt{live}, \texttt{dead}\}) \to \{\texttt{live}, \texttt{dead}\}$$
that takes whether v is `live` or `dead` as its first coordinate and a function from $\{-1, 0, 1\} \times \{-1, 0, 1\} - \{(0, 0)\}$ to $\{\texttt{live}, \texttt{dead}\}$ that says whether each of its eight neighbors is `live` or `dead` as a second coordinate, then executes the rules from Conway's Game of Life to determine whether it should be live or dead in the next time step.

Solution to Exercise 4.76.

We compute $[q, r]$ for various values of $q, r \in \mathbf{Poly}$ using (4.75).

1. If $q \coloneqq 0$, then $q(1) \cong 0$, so $[q, r]$ is an empty product. Hence $[q, r] \cong 1$.
2. If $q \coloneqq 1$, then $q(1) \cong 1$ and $q[1] \cong 0$, so $[q, r] \cong r \circ (0y) \cong r(0)$.
3. If $q \coloneqq y$, then $q(1) \cong 1$ and $q[1] \cong 1$, so $[q, r] \cong r \circ (1y) \cong r$.
4. If $q \coloneqq A$ 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 $q \coloneqq Ay$ 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 $q \coloneqq y^2 + 2y$ and $r \coloneqq 2y^3 + 3$, 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 4.77.

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 (4.75),

$$[p_1 + p_2, q] \cong \left(\prod_{i \in p_1(1)} q \circ (p_1[i]y)\right)\left(\prod_{i \in p_2(1)} q \circ (p_2[i]y)\right) \cong [p_1, q] \times [p_2, q].$$

Solution to Exercise 4.78.

We may compute

$$\begin{aligned}
[q, r] &\cong \prod_{j \in q(1)} r \circ (q[j]y) && (4.75) \\
&\cong \prod_{j \in q(1)} \sum_{k \in r(1)} (q[j]y)^{r[k]} && (\text{Replacing each } y \text{ in } r \text{ by } q[j]y) \\
&\cong \sum_{f_1 \colon q(1) \to r(1)} \prod_{j \in q(1)} (q[j]y)^{r[f_1(j)]} && (1.30) \\
&\cong \sum_{f_1 \colon q(1) \to r(1)} \left(\prod_{j \in q(1)} q[j]^{r[f_1(j)]}\right)\left(\prod_{j \in q(1)} y^{r[f_1(j)]}\right) \\
&\cong \sum_{f_1 \colon q(1) \to r(1)} \sum_{f^\sharp \in \prod_{j \in q(1)} q[j]^{r[f_1(j)]}} y^{\sum_{j \in q(1)} r[f_1(j)]} \\
&\cong \sum_{f \colon q \to r} y^{\sum_{j \in q(1)} r[f_1(j)]}. && (3.7)
\end{aligned}$$

Solution to Exercise 4.80.

We verify (3.24) as follows:

$$\begin{aligned}
[p, y] \otimes p &\cong \left(\sum_{f \colon p \to y} y^{\sum_{i \in p(1)} y[f_1 i]}\right) \otimes p && (4.79) \\
&\cong \sum_{f \in \Gamma(p)} y^{p(1)} \otimes \sum_{i \in p(1)} y^{p[i]} \\
&\cong \sum_{f \in \Gamma(p)} \sum_{i \in p(1)} y^{p(1) \times p[i]} && (3.66) \\
&\cong \sum_{f \in \prod_{i \in p(1)} p[i]} \sum_{i \in p(1)} y^{p(1) \times p[i]}. && (3.35)
\end{aligned}$$

Solution to Exercise 4.83.

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

$$\begin{aligned} [p, y] &\cong \sum_{f \colon p \to y} y^{\sum_{i \in p(1)} y[f_1 i]} && (4.79) \\ &\cong \sum_{f \in \Gamma(p)} y^{\sum_{i \in p(1)} 1} \\ &\cong \Gamma(p) y^{p(1)}. \end{aligned}$$

Solution to Exercise 4.84.

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 (4.82) twice, we have that

$$[[p, y], y] \cong \Gamma\left(\Gamma(p) y^{p(1)}\right) y^{\Gamma(p)}.$$

By (3.35),

$$\Gamma\left(\Gamma(p) y^{p(1)}\right) \cong \prod_{\gamma \in \Gamma(p)} p(1) \cong p(1)^{\Gamma(p)}.$$

Hence taking $[[p, y], y] \cong p$ and rewriting the left hand side using the isomorphisms above yields

$$p(1)^{\Gamma(p)} y^{\Gamma(p)} \cong p. \tag{4.94}$$

In particular, p is a monomial, so we can write $p \coloneqq I y^A$ for some $I, A \in \mathbf{Set}$. Then $p(1) \cong I$ and (3.35) tells us that $\Gamma(p) \cong A^I$. Equating the direction-sets on either side of (4.94) yields $A^I \cong A$; then equating position-sets gives $I^A \cong I^{A^I} \cong I$.

We conclude with some elementary set theory. If either one of I or A were (isomorphic to) 1, then p would be either linear or representable, and we would be done. Meanwhile, if either one of I or A were 0, then the other would be 1, and we would again be done. Otherwise, $|A|, |B| \geq 2$. But by Cantor's theorem,

$$|I| < \left|2^I\right| \leq \left|A^I\right| = |A| \qquad \text{and} \qquad |A| < \left|2^A\right| \leq \left|I^A\right| = |I|,$$

a contradiction.

Solution to Exercise 4.87.

The isomorphism $\mathbf{Poly}(p, q) \cong [p, q](1)$ follows directly from Exercise 4.78 when both sides are applied to 1. Alternatively, we can apply (4.86). Since $p \cong y \otimes p$, we have that

$$\begin{aligned} \mathbf{Poly}(p, q) &\cong \mathbf{Poly}(y \otimes p, q) \\ &\cong \mathbf{Poly}(y, [p, q]) && (4.86) \\ &\cong [p, q](1). && \text{(Yoneda lemma)} \end{aligned}$$

Solution to Exercise 4.89.

To obtain the evaluation lens $\mathrm{eval} \colon [q, r] \otimes q \to r$, we need to send the identity lens on $[q, r]$ leftward through the natural isomorphism

$$\mathbf{Poly}([q, r] \otimes q, r) \cong \mathbf{Poly}([q, r], [q, r])$$

To do so, we can start from the identity lens on $[q, r]$ and work our way along a chain of natural isomorphisms from $\mathbf{Poly}([q, r], [q, r])$ until we get to $\mathbf{Poly}([q, r] \otimes q, r)$. To start, Exercise 4.78 implies that

$$\begin{aligned}\mathbf{Poly}([q, r], [q, r]) &\cong \mathbf{Poly}\left(\sum_{f\colon q \to r} \prod_{i' \in q(1)} y^{r[f_1 i']}, \prod_{i \in q(1)} \sum_{j \in r(1)} (q[i]y)^{r[j]}\right) \\ &\cong \prod_{f\colon q \to r} \prod_{i \in q(1)} \mathbf{Poly}\left(\prod_{i' \in q(1)} y^{r[f_1 i']}, \sum_{j \in r(1)} (q[i]y)^{r[j]}\right),\end{aligned}$$

where the second isomorphism follows from the universal properties of products and coproducts. In particular, under this isomorphism, the identity lens on $[q, r]$ corresponds to a collection of lenses, namely for each $f\colon q \to r$ and each $i \in q(1)$ the composite

$$\prod_{i' \in q(1)} y^{r[f_1 i']} \to y^{r[f_1 i]} \to \sum_{g\colon r[f_1 i] \to q[i]} y^{r[f_1 i]} \cong (q[i]y)^{r[f_1 i]} \to \sum_{j \in r(1)} (q[i]y)^{r[j]}$$

of the canonical projection with index $i' \coloneqq i$, the canonical inclusion with index $g \coloneqq f_i^\sharp$, and the canonical inclusion with index $j \coloneqq f_1 i$. On positions, this lens picks out the position of $\sum_{j \in r(1)} (q[i]y)^{r[j]}$ corresponding to $j = f_1 i \in r(1)$ and $f_i^\sharp\colon r[f_1 i] \to q[i]$; on directions, the lens is the canonical inclusion $r[f_1 i] \to \sum_{i' \in q(1)} r[f_1 i']$ with index $i' = i$. By the Yoneda lemma, we can reinterpret each of these lenses as a lens

$$y^{q[i] \times \sum_{i' \in q(1)} r[f_1 i']} \to \sum_{j \in r(1)} y^{r[j]} \cong r$$

that, on positions, picks out the position $f_1 i \in r(1)$ of r and, on directions, is the map $r[f_1 i] \to q[i] \times \sum_{i' \in q(1)} r[f_1 i']$ induced by the universal property of products applied to the map $f_i^\sharp\colon r[f_1 i] \to q[i]$ and the inclusion $r[f_1 i] \to \sum_{i' \in q(1)} r[f_1 i']$. Then by the universal property of coproducts, this collection of lenses induces a single lens $\mathrm{eval}\colon [q, r] \otimes q \to r$ that sends each position $f\colon q \to r$ of $[q, r]$ and position $i \in q(1)$ of q to the position $f_1 i$ of r, with the same behavior on directions as the corresponding lens described previously.

Solution to Exercise 4.90.

1. Given a set S, Example 4.81 shows that

$$[Sy, y] \otimes (Sy) \cong y^S \otimes (Sy) \cong Sy^S,$$

so by setting $q \coloneqq Sy$ and $r \coloneqq y$ in (4.88), we obtain an evaluation lens $\mathrm{eval}\colon Sy^S \to y$. By the solution to Exercise 4.89, given a position $s \in S$ of Sy^S, the evaluation lens 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 lenses in (4.91) from Example 4.70 as the parallel product of identity lenses and evaluation lenses. By the solution to Exercise 4.89, the evaluation lens $[Fy, y^F] \otimes Fy \to y^F$ is a lens from

$$[Fy, y^F] \otimes Fy \cong F\left(\sum_{f\colon Fy \to y^F} \prod_{i \in F} y^F\right) \cong Fy^{FF}$$

to y^F that is uniquely determined on positions and has the on-directions map $FF \to FF$ given by the identity. Then we can verify that κ_{11} is equivalent to the parallel product of this evaluation lens with itself. We can define κ_{12} and κ_{21} to be the parallel product of this evaluation lens with the identity on y^F, while κ_{22} is the parallel product of the identity on y^F with itself.

Chapter 5

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 check back here when needed. Better yet might be to gently leaf through this chapter, to see how well-behaved and versatile the category **Poly** is.

5.1 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) representable (or pure power) polynomials $1, y, y^2, y^A$; and
d) monomials $0, A, y, 2y^3, By^A$.

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 5.2 and 5.3. The third puts a copy of $\mathbf{Set}^{\mathrm{op}}$ inside **Poly**: it is the Yoneda embedding that we saw way back in Exercise 1.12. Finally, the fourth puts a copy of bimorphic lenses inside **Poly**, as we saw in Example 3.41.

Exercise 5.1 (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 $p \circ q$? (We have not discussed this yet, so feel free to skip it.) ◊

Proposition 5.2. There is a fully faithful functor **Set** $\to$ **Poly** sending $A \mapsto Ay^0 = A$.

Proof. By (3.8), a lens $f\colon Ay^0 \to By^0$ 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 function between sets $A \to B$. □

Proposition 5.3. There is a fully faithful functor **Set** $\to$ **Poly** sending $A \mapsto Ay$.

Proof. By (3.8), a lens $f\colon Ay^1 \to By^1$ 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 function between sets $A \to B$. □

Theorem 5.4. **Poly** has an adjoint quadruple with **Set**:

$$\begin{tikzcd}[column sep=huge]
\mathbf{Set}
\arrow[r, shift left=0.6em, "A" description]
\arrow[r, shift right=1.8em, "Ay"']
& \mathbf{Poly}
\arrow[l, shift right=1.8em, "p(0)"']
\arrow[l, shift left=0.6em, "p(1)" description]
\end{tikzcd} \tag{5.5}$$

where the functors have been labeled by where they send $A \in$ **Set** and $p \in$ **Poly**.

Both rightward functors **Set** $\to$ **Poly** are fully faithful.

Proof. For any set A, there is a functor **Poly** $\to$ **Set** given by sending p to $p(A)$; by the Yoneda lemma, it is the functor $\mathbf{Poly}(y^A, -)$. This, together with Propositions 5.2 and 5.3, 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 (3.7) for computing general hom-sets in **Poly**; we leave the details to the reader in Exercise 5.6. □

Exercise 5.6 (Solution here). Here we prove the remainder of Theorem 5.4 using (3.7):

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 5.7 (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 5.4 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 5.10. 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 (5.5)—is special to $A = 0, 1$.

We also have a copower-hom-power two-variable adjunction between **Poly**, **Set**, and **Poly**.

Proposition 5.8. There is a two-variable adjunction between **Poly**, **Set**, and **Poly**:

$$\mathbf{Poly}(Ap, q) \cong \mathbf{Set}(A, \mathbf{Poly}(p, q)) \cong \mathbf{Poly}(p, q^A). \tag{5.9}$$

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 (5.9), we obtain the following using the Yoneda lemma.

Corollary 5.10. For any set B there is an adjunction

$$\begin{tikzcd} \mathbf{Set} \arrow[r, shift left=2, "Ay^B"] \arrow[r, phantom, "\Rightarrow"] & \mathbf{Poly} \arrow[l, shift left=2, "q(B)"] \end{tikzcd}$$

where the functors are labeled by where they send $q \in \mathbf{Poly}$ and $A \in \mathbf{Set}$.

Exercise 5.11 (Solution here). Prove Corollary 5.10 from Proposition 5.8. ◊

Proposition 5.12. The Yoneda embedding $A \mapsto y^A$ has a left adjoint

$$\begin{tikzcd} \mathbf{Set}^{\mathrm{op}} \arrow[r, shift left=2, "y^-"] \arrow[r, phantom, "\Leftarrow"] & \mathbf{Poly} \arrow[l, shift left=2, "\Gamma"] \end{tikzcd}$$

where $\Gamma(p) := \mathbf{Poly}(p, y) \cong \prod_{i \in p(1)} p[i]$, as in (3.37) and (3.35). That is, there is a natural isomorphism

$$\mathbf{Poly}(p, y^A) \cong \mathbf{Set}(A, \Gamma(p)). \tag{5.13}$$

Proof. By (3.7), 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 (3.35). □

Exercise 5.14 (Solution here). Prove Proposition 5.12 from Proposition 5.8. ◊

Corollary 5.15 (Principal monomial)**.** There is an adjunction

$$\mathbf{Poly} \underset{Ay^B}{\overset{(p(1),\Gamma(p))}{\underset{\longleftarrow}{\overset{\longrightarrow}{\Rightarrow}}}} \mathbf{Set} \times \mathbf{Set}^{\mathrm{op}}$$

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

$$\mathbf{Poly}(p, Ay^B) \cong \mathbf{Set}(p(1), A) \times \mathbf{Set}(B, \Gamma(p)). \tag{5.16}$$

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 5.6 #2 and (5.13). □

Exercise 5.17 (Solution here). Use (5.16) together with (4.82) and (4.86) to find an alternative proof for Proposition 4.54, 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 $p, q \in \mathbf{Poly}$. ◊

5.2 Epi-mono factorization of lenses

Proposition 5.18. Let $f : p \to q$ be a lens 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 5.4), 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 lenses $g, h : y^A \rightrightarrows p$ 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 $g, h\colon r \rightrightarrows p$ be two lenses 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 5.19. 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 $\mathbb{N}y^{\mathbb{N}} \to y$ from Example 4.6.

On positions, we have $f_1 i = i$ for all $i \in \mathsf{k}$. On directions, for any $i \in \mathsf{k}$, we have $f_i^\sharp(n) = n \mod k$ for all $n \in \mathbb{N}$.

Exercise 5.20 (Solution here). In Example 5.19, we gave a lens $\mathsf{12}y^{\mathsf{12}} \to \mathbb{N}y^{\mathbb{N}}$. 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 5.21. Let $f\colon p \to q$ be a lens 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

$$f_j^\flat\colon q[j] \to \prod_{\substack{i \in p(1),\\ f_1 i = j}} p[i]$$

from (3.20) 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 5.4), 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 $f_j^\flat$ is a monomorphism. Suppose we are given a set A and a pair of functions $g', h'\colon A \rightrightarrows q[j]$ with $g' \mathbin{⨾} f_j^\flat = h' \mathbin{⨾} f_j^\flat$. They can be identified with lenses $g, h\colon q \rightrightarrows y^A + 1$, 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 lenses $g, h\colon q \rightrightarrows y^A + 1$ 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 $g_j^\sharp \coloneqq g'$ and $h_j^\sharp \coloneqq h'$. 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 lenses such that $f \mathbin{⨾} g = f \mathbin{⨾} h$. Then $f_1 \mathbin{⨾} g_1 = f_1 \mathbin{⨾} h_1$, which implies $g_1 = h_1$; we'll consider g_1 the default representation. We also have that $g^\sharp_{f_1 i} \mathbin{⨾} f_i^\sharp = h^\sharp_{f_1 i} \mathbin{⨾} f_i^\sharp$ for any $i \in p(1)$. It follows that, for any $j \in q(1)$, the two composites

$$r[g_1 j] \overset{g_j^\sharp}{\underset{h_j^\sharp}{\rightrightarrows}} q[j] \xrightarrow{f_j^\flat} \prod_{\substack{i \in p(1),\\ f_1 i = j}} p[i]$$

are equal, which implies that $g_j^\sharp = h_j^\sharp$ as desired. □

Exercise 5.22 (Solution here). Show that the only way for a lens $p \to y$ to *not* be an epimorphism is when $p = 0$. ◊

Exercise 5.23 (Solution here). Let A and B be sets and AB their product. Find an epimorphism $y^A + y^B \twoheadrightarrow y^{AB}$. ◊

Exercise 5.24 (Solution here). Suppose a lens $f : p \to q$ 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 5.25 (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 $f = e \mathbin{⨾} m$;
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 5.26 (Epi-mono factorization in **Set**). The category **Set** has epi-mono factorization: a function $f: X \to Y$ can be uniquely factored into an epimorphism (surjection) e followed by a monomorphism (injection) i, as follows. The epimorphism $e: X \to f(X)$ 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 $i: f(X) \to Y$ 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 5.27. **Poly** has epi-mono factorization.

Proof. Take an arbitrary lens $\varphi: p \to q$. It suffices to show that there exists a unique polynomial r equipped with an epimorphism $\epsilon: p \to r$ and a monomorphism $\mu: r \to q$ such that $\varphi = \epsilon \mathbin{⨟} \mu$.

On positions, we must have $\varphi_1 = \epsilon_1 \mathbin{⨟} \mu_1$, with μ_1 a monomorphism and ϵ_1 an epimorphism per Propositions 5.18 and 5.21. By Example 5.26, since **Set** has epi-mono factorization, such $r(1), \epsilon_1$, and μ_1 uniquely exist. In particular, we must have that $r(1) \cong \varphi_1(p(1))$, that $\epsilon_1: p(1) \to \varphi_1(p(1))$ is the corestriction of φ_1 sending $i \mapsto \varphi_1(i)$ for each p-position i, and that $\mu_1: \varphi_1(p(1)) \to q(1)$ 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

$$\begin{tikzcd} q[\varphi_1(i)] \arrow[r, "\mu^\sharp_{\varphi_1(i)}"] \arrow[dr, "\varphi^\sharp_i"'] & r[\varphi_1(i)] \arrow[d, "\epsilon^\sharp_i"] \\ & p[i] \end{tikzcd}$$

commutes—or, equivalently, for every $j \in r(1) \cong \varphi_1(p(1))$,

$$\begin{tikzcd} q[j] \arrow[r, "\mu^\sharp_j"] \arrow[dr, "\varphi^\flat_j"'] & r[j] \arrow[d, "\epsilon^\flat_j"] \\ & \prod\limits_{\substack{i \in p(1), \\ \varphi_1(i) = j}} p[i] \end{tikzcd}$$

commutes (here $\varphi^\flat_j$ and $\epsilon^\flat_j$ are the induced functions from (3.20)), with $\mu^\sharp_j$ an epimorphism and $\epsilon^\flat_j$ a monomorphism per Propositions 5.18 and 5.21. 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. $\square$

5.3 Cartesian closure

We have already seen in Section 4.5 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

$$r^q \coloneqq \prod_{j \in q(1)} r \circ (y + q[j]) \tag{5.28}$$

where $\circ$ denotes composition.

Before proving that this really is an exponential in **Poly**, which we do in Theorem 5.31, we first get some practice with it.

Example 5.29. Let A be a set. We've been writing the polynomial Ay^0 simply as A, so it better be true that there is an isomorphism

$$y^A \cong y^{Ay^0}$$

in order for the notation to be consistent. Luckily, this is true. By (5.28), we have

$$y^{Ay^0} = \prod_{a \in A} y \circ (y + 0) \cong y^A$$

Exercise 5.30 (Solution here). Compute the following exponentials in **Poly** using (5.28):

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 5.31. 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 (5.28).

Proof. We have the following chain of natural isomorphisms:

$$\mathbf{Poly}(p, r^q) \cong \mathbf{Poly}\Big(p, \prod_{j \in q(1)} r \circ (y + q[j])\Big) \tag{5.28}$$

$$
\begin{aligned}
&\cong \prod_{i\in p(1)} \prod_{j\in q(1)} \mathbf{Poly}\big(y^{p[i]}, r \circ (y + q[j])\big) \\
&\hspace{16em} \text{(Universal property of (co)products)} \\
&\cong \prod_{i\in p(1)} \prod_{j\in q(1)} r \circ (p[i] + q[j]) && \text{(Yoneda lemma)} \\
&\cong \prod_{i\in p(1)} \prod_{j\in q(1)} \sum_{k\in r(1)} (p[i] + q[j])^{r[k]} \\
&\cong \prod_{(i,j)\in(p\times q)(1)} \sum_{k\in r(1)} (p\times q)[(i,j)]^{r[k]} && (3.61) \\
&\cong \mathbf{Poly}(p\times q, r). && (3.7)
\end{aligned}
$$

□

Exercise 5.32 (Solution here). Use Theorem 5.31 to show that for any polynomials p, q, there is a canonical evaluation lens

$$\text{eval}\colon p^q \times q \to p.$$

◊

5.4 Limits and colimits of polynomials

We have already seen that **Poly** has all coproducts (Proposition 3.3) and products (Proposition 3.56). We will now see that **Poly** has all small limits and colimits.

Theorem 5.33. 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 3.56, 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 lens $e\colon p' \to p$. One can check that p' together with e satisfies the universal property of the equalizer of f and g; see Exercise 5.34. □

[1] If we're being precise, a "(co)equalizer" is an object equipped with a morphism, but we will use the term to refer to either just the object or just the morphism when the context is clear.

Exercise 5.34 (Solution here). Complete the proof of Theorem 5.33 as follows:

1. We said that p' comes equipped with a lens $e\colon p' \to p$; 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 5.35 (Computing general limits in **Poly**). The proof of Theorem 5.33 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

$$\left(\lim_{j\in\mathcal{J}} p_j\right)(1) \cong \lim_{j\in\mathcal{J}} p_j(1), \tag{5.36}$$

equipped with a canonical projection π_j to each $p_j(1)$, and whose direction-set for each position i is

$$\left(\lim_{j\in\mathcal{J}} p_j\right)[i] \cong \operatorname*{colim}_{j\in\mathcal{J}^{\mathrm{op}}} p_j[\pi_j(i)]. \tag{5.37}$$

This notation obscures what is occuring on lenses, but in particular, each lens $\varphi\colon p_j \to p_{j'}$ in the diagram p_- induces an on-positions function $\varphi_1\colon p_j(1) \to p_{j'}(1)$ in the diagram whose limit we take in (5.36) and, for every position i of the limit, an on-directions function $\varphi^\sharp_{\pi_j(i)}\colon p_{j'}[\pi_{j'}(i)] \to p_j[\pi_j(i)]$ in the diagram whose colimit we take in (5.37). (Note that, by the definition of a limit, $\varphi_1(\pi_j(i)) = \pi_{j'}(i)$.)

We have seen (5.36) and (5.37) 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 (5.36) and (5.37) to be true for equalizers in the proof of Theorem 5.33. It follows from the construction of any limit as an equalizer of products that it is true for arbitrary limits.

Example 5.38 (Pullbacks in **Poly**). Given $q, q', r \in \mathbf{Poly}$ and lenses $q \xrightarrow{f} r \xleftarrow{f'} q'$, 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 give the lenses g and g' on positions and on directions:

$$
\begin{tikzcd}
p(1) \arrow[r, "g'_1"] \arrow[d, "g_1"'] \arrow[dr, phantom, "\lrcorner", very near start] & q'(1) \arrow[d, "f'_1"] \\
q(1) \arrow[r, "f_1"'] & r(1)
\end{tikzcd}
\qquad \text{and} \qquad
\begin{tikzcd}
p[(i,i')] \arrow[dr, phantom, "\lrcorner", very near start] & q'[i'] \arrow[l, "(g')^\sharp_{(i,i')}"'] \\
q[i] \arrow[u, "g^\sharp_{(i,i')}"] & r[f_1(i)] \arrow[l, "f^\sharp_i"] \arrow[u, "(f')^\sharp_{i'}"']
\end{tikzcd}
\tag{5.39}
$$

Exercise 5.40 (Solution here). Let p be any polynomial.

1. There is a canonical choice of lens $\eta\colon p \to p(1)$; what is it?
2. Given an element $i \in p(1)$, i.e. a function (or lens 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 lenses $f\colon p_i \to 1$ and $g\colon p_i \to p$? $\lozenge$

Exercise 5.41 (Solution here). Let $q \coloneqq y^2 + y$, $q' \coloneqq 2y^3 + y^2$, and $r \coloneqq y + 1$.

1. Choose lenses $f\colon q \to r$ and $f'\colon q' \to r$ and write them down.
2. Find the pullback of $q \xrightarrow{f} r \xleftarrow{f'} q'$. $\lozenge$

Exercise 5.42 (Solution here). An alternative way to prove Theorem 5.33 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 5.33. 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 5.33.

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 $\mathbf{Set}^{\mathbf{Set}}$. ◊

Theorem 5.43. 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 3.3, 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 $G \colon \boxed{\bullet \rightrightarrows \bullet} \to \mathbf{Set}$ 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 $F \colon (\int G_c)^{\mathrm{op}} \to \mathbf{Set}$ 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 5.44. □

Exercise 5.44 (Solution here). Complete the proof of Theorem 5.43 as follows:

1. Provide a lens $g \colon q \to q'$.
2. Show that $s \mathbin{⨟} g = t \mathbin{⨟} g$.
3. Show that g is a coequalizer of the pair s, t. ◊

Example 5.45. 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 $\mathbf{Set}^{\mathbf{Set}}$). We saw in Exercise 5.42 that in the case of limits, these yield the same result. So, too, in the case of coproducts, per Proposition 3.3.

But in the case of general colimits, there are diagrams that yield different results: by the co-Yoneda lemma, *every* functor $\mathbf{Set} \to \mathbf{Set}$—even those that are not polynomials—can be written as the colimit of representable functors in $\mathbf{Set}^{\mathbf{Set}}$, 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

$$y^2 \rightrightarrows y. \tag{5.46}$$

According to Theorem 5.43, the colimit of (5.46) 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 (5.46) in **Poly** is the constant functor $1y^0 \cong 1$.

But as functors, by Proposition 1.37, the colimit of (5.46) 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 5.47 (Solution here). By Theorem 5.4, for any polynomial p, there are canonical lenses involving positions and global sections:

$$\epsilon\colon p(1)y \to p \qquad \text{and} \qquad \eta\colon p \to y^{\Gamma(p)}.$$

1. Characterize the behavior of the canonical lens $\epsilon\colon p(1)y \to p$.
2. Characterize the behavior of the canonical lens $\eta\colon p \to y^{\Gamma(p)}$.
3. Show that the following is a pushout in **Poly**:

$$\begin{tikzcd} p(1)y \arrow[r, "!"] \arrow[d, "\epsilon"'] \arrow[dr, phantom, "\ulcorner"] & y \arrow[d, "!"] \\ p \arrow[r, "\eta"'] & y^{\Gamma(p)} \end{tikzcd} \tag{5.48}$$

◊

Proposition 5.49. For polynomials p, q, the following is a pushout:

$$\begin{tikzcd} p(1)y \otimes q(1)y \arrow[r] \arrow[d] \arrow[dr, phantom, "\ulcorner"] & p(1)y \otimes q \arrow[d] \\ p \otimes q(1)y \arrow[r] & p \otimes q \end{tikzcd}$$

Proof. All the lenses 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

$$\begin{tikzcd} y \arrow[r] \arrow[d] \arrow[dr, phantom, "\ulcorner"] & y^{q[j]} \arrow[d] \\ y^{p[i]} \arrow[r] & y^{p[i]\times q[j]} \end{tikzcd} \qquad \begin{tikzcd} 1 & q[j] \arrow[l] \\ p[i] \arrow[u] & p[i] \times q[j] \arrow[l] \arrow[u] \arrow[ul, phantom, "\ulcorner"] \end{tikzcd}$$

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 5.12. $\square$

This means that to give a lens $\varphi\colon p \otimes q \to r$, it suffices to give two lenses $\varphi_p\colon p \otimes q(1)y \to r$ and $\varphi_q\colon p(1)y \otimes q \to r$ that agree on positions. The lens φ_p says how information about q's position is transferred to p, and the lens φ_q says how information about p's position is transferred to q.

Corollary 5.50. 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 5.49. $\square$

5.5 Vertical-cartesian factorization of lenses

Aside from epi-mono factorization, there is another factorization system on **Poly** that will show up frequently.

Definition 5.51 (Vertical and cartesian lenses)**.** 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 5.52. Vertical and cartesian lenses form a factorization system of **Poly**.

Proof. It is easy to check that isomorphisms are both vertical and cartesian, and that vertical and cartesian lenses are each closed under composition. It remains to show that every lens in **Poly** can be uniquely (up to unique isomorphism) factored as a vertical lens composed with a cartesian lens.

Recall from (3.8) that a lens in **Poly** can be written as to the left; we can thus rewrite it as to the right:

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

We can see that the intermediary object $\sum_{i \in p(1)} y^{q[f_1 i]}$ is unique up to unique isomorphism. □

Proposition 5.53. Vertical lenses 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 lenses 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 5.54 (Solution here). Give an example of polynomials p, q, r and lenses $p \xrightarrow{f} q \xrightarrow{g} r$ such that f and $f \mathbin{⨾} g$ are cartesian but g is not. ◊

Here is an alternative characterization of a cartesian lens in **Poly**. Recall from Exercise 3.25 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:

$$
\begin{tikzcd}
\dot{p}(1) \arrow[d, "\pi_p"'] & \bullet \arrow[l, "f^\sharp"'] \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \dot{q}(1) \arrow[d, "\pi_q"] \\
p(1) \arrow[r, equal] & p(1) \arrow[r, "f_1"'] & q(1)
\end{tikzcd}
\tag{5.55}
$$

Here, the pullback denoted by the dot $\bullet$ 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 5.56 (Solution here). Show that a lens $f \colon p \to q$ in **Poly** is cartesian if and only if the square on the left hand side of (5.55) is also a pullback:

$$\begin{tikzcd}
\dot{p}(1) \arrow[d, "\pi_p"'] & \bullet \arrow[l, "f^\sharp"'] \arrow[r] \arrow[d] \arrow[dl, phantom, very near start, "\llcorner"] \arrow[dr, phantom, very near start, "\lrcorner"] & \dot{q}(1) \arrow[d, "\pi_q"] \\
p(1) \arrow[r, equal] & p(1) \arrow[r, "f_1"'] & q(1)
\end{tikzcd}$$

◊

Exercise 5.57 (Solution here). Is the pushout of a cartesian lens 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 5.58 (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 5.59. Let $f: p \to q$ be a morphism in **Poly**. The following are equivalent:

1. viewed as a lens, f is cartesian in the sense of Definition 5.51: 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 (5.55) 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 5.58: for any sets A, B and function $h: A \to B$, the naturality square

$$\begin{tikzcd}
p(A) \arrow[r, "f_A"] \arrow[d, "p(h)"'] \arrow[dr, phantom, very near start, "\lrcorner"] & q(A) \arrow[d, "q(h)"] \\
p(B) \arrow[r, "f_B"'] & q(B)
\end{tikzcd} \tag{5.60}$$

   is a pullback.

Proof. We already showed that the first two are equivalent in Exercise 5.56, and we will complete this proof in Exercise 5.61. □

Exercise 5.61 (Solution here). In this exercise, you will complete the proof of Proposition 5.59.

First, we will show that $1 \Rightarrow 3$. In the following, let $f: p \to q$ be a cartesian lens in **Poly** and $h: A \to B$ be a function.

1. Using Proposition 3.44 to translate f from a lens in **Poly** to a natural transformation and Proposition 2.10 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 (5.60), hence proving $1 \Rightarrow 3$.

Next, we show that $3 \Rightarrow 1$. In the following, let $f: p \to q$ be a lens in **Poly** that is cartesian when viewed as a natural transformation, so that (5.60) is a pullback for any function $h: A \to B$. Also fix $i \in p(1)$.

3. Show that the diagram

$$\begin{tikzcd} 1 \arrow[r, "{(f_1 i, \mathrm{id}_{q[f_1 i]})}"] \arrow[d, "{(i, \mathrm{id}_{p[i]})}"'] \arrow[dr, phantom, "\lrcorner", very near start] & q(q[f_1 i]) \arrow[d, "q(f_i^\sharp)"] \\ p(p[i]) \arrow[r, "f_{p[i]}"'] & q(p[i]) \end{tikzcd} \tag{5.62}$$

commutes. Hint: Use Proposition 2.10, Proposition 3.44, and/or Corollary 3.47.

4. Apply the universal property of the pullback (5.60) to the diagram (5.62) 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 5.63. 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 lens by passing it through the functor $\mathbf{Poly} \xrightarrow{p(1)} \mathbf{Set}$ from Theorem 5.4. 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 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 lens $(f \times g)^\sharp_{(i,j)}$ (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 5.64. Pullbacks preserve vertical (resp. cartesian) lenses. In other words, if $f\colon p \to q$ is a lens and $g\colon q' \to q$ a vertical (resp. cartesian) lens, 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 5.38, since the pullback (resp. pushout) of an isomorphism is an isomorphism. □

5.6 Monoidal ∗-bifibration over Set

We conclude this chapter by showing that the functor $p \mapsto p(1)$ has special properties that make it what [Shu08] refers to as a *monoidal ∗-bifibration*. Roughly speaking, 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. The material in this section is even more technical than the rest of this chapter, and we won't use it again in the book, so the reader may wish to skip to Part II.

As an 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 lens between constant polynomials. From f, we can obtain a new polynomial f^*p with position-set A via a pullback

$$\begin{tikzcd} f^*p \arrow[r, "\text{cart}"] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & p \arrow[d, "\eta_p"] \\ A \arrow[r, "f"'] & p(1) \end{tikzcd} \tag{5.65}$$

Here η_p is the unit of the adjunction $\begin{tikzcd} \mathbf{Set} \arrow[r, shift left=2, "A"] \arrow[r, phantom, "\Leftarrow"] & \mathbf{Poly} \arrow[l, shift left=2, "p(1)"] \end{tikzcd}$; it is a vertical lens. We could evaluate this pullback using Example 5.38. Alternatively, we can use Proposition 5.64 to deduce that the top lens $f^*p \to p$ (which we presciently labeled cart) is cartesian like f and that the left lens $f^*p \to A$ 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 5.72 and Theorem 5.73, but first we need the following definitions and a result about cartesian lenses.

Definition 5.66 (Slice category)**.** Given an object c in a category $\mathcal{C}$, the *slice category* of $\mathcal{C}$ over c, denoted $\mathcal{C}/c$, is the category whose objects are morphisms in $\mathcal{C}$ with codomain c and whose morphisms are commutative triangles in $\mathcal{C}$.

Definition 5.67 (Exponentiable morphism)**.** 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 5.68. Cartesian lenses 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:

$$\begin{tikzcd} \mathbf{Poly}/p \arrow[r, shift right=2, "f_*"'] \arrow[r, phantom, "\Leftarrow"] & \mathbf{Poly}/q \arrow[l, shift right=2, "f^*"'] \end{tikzcd}$$

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:

$$f_*p' \coloneqq \sum_{j\in q(1)} \sum_{i'\in \prod\limits_{i\in f_1^{-1}(j)} e_1^{-1}(i)} y^{q[j]/\sum_{i\in f_1^{-1}(j)} p'[i'(i)]} \tag{5.69}$$

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

$$\begin{aligned}
(\mathbf{Poly}/p)(f^*q', p') &\cong \prod_{i\in p(1)} \prod_{\{j'\in q'(1) \mid g_1(j')=f_1 i\}} \sum_{\{i'\in p'(1) \mid e_1(i')=i\}} (p[i]/\mathbf{Set})(p'[i'], p[i] +_{q[f(i)]} q'[j']) \\
&\cong \prod_{i\in p(1)} \prod_{\{j'\in q'(1) \mid g_1(j')=f_1 i\}} \sum_{\{i'\in p'(1) \mid e_1(i')=i\}} (q[f(i)]/\mathbf{Set})(p'[i'], q'[j']) \\
&\cong \prod_{j\in q(1)} \prod_{\{j'\in q'(1) \mid g_1(j')=j\}} \prod_{\{i\in p(1) \mid f_1 i=j\}} \sum_{\{i'\in p'(1) \mid e_1(i')=i\}} (q[j]/\mathbf{Set})(p'[i'], q'[j']) \\
&\cong \prod_{j\in q(1)} \prod_{\{j'\in q'(1) \mid g_1(j')=j\}} \sum_{i'\in \prod_{i\in f_1^{-1}(j)} e_1^{-1}(i)} \prod_{i\in f_1^{-1}(j)} (q[j]/\mathbf{Set})(p'[i'(i)], q'[j']) \\
&\cong \prod_{j\in q(1)} \prod_{\{j'\in q'(1) \mid g_1(j')=j\}} \sum_{i'\in \prod_{i\in f_1^{-1}(j)} e_1^{-1}(i)} (q[j]/\mathbf{Set})\Big(\sum_{i\in f_1^{-1}(j)} p'[i'(i)], q'[j']\Big) \\
&\cong (\mathbf{Poly}/q)(q', f_*p')
\end{aligned}$$

□

Example 5.70. Let $p \coloneqq 2y^2$, $q \coloneqq y^2 + y^0$, and $f\colon p \to q$ the unique cartesian lens between them. Then for any $e\colon p' \to p$ over p, (5.69) 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 (5.69) 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 5.71 (Solution here). Prove that the unique lens $f\colon y \to 1$ 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 5.72 (Base change)**.** For any function $f\colon A \to B$, pullback f^* along f induces a functor $B.\mathbf{Poly} \to A.\mathbf{Poly}$, which we also denote f^*.

Proof. This follows from (5.39) with $q := A$ and $r := B$, since pullback of an iso is an iso. □

Theorem 5.73. For any function $f\colon A \to B$, the pullback functor f^* has both a left and a right adjoint

$$\begin{tikzcd} A.\mathbf{Poly} \arrow[r, bend left, "f_!"] \arrow[r, bend right, "f_*"'] & B.\mathbf{Poly} \arrow[l, "f^*" description] \end{tikzcd} \qquad \text{with } f_! \Rightarrow f^*, \; f^* \Leftarrow f_* \tag{5.74}$$

Moreover ⊗ 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:

$$f_! p \cong \sum_{b \in B} y^{\prod_{a \mapsto b} p[a]} \qquad \text{and} \qquad f_* p \cong \sum_{b \in B} y^{\sum_{a \mapsto b} p[a]} \tag{5.75}$$

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 the fact that **Poly** is equivalent to the Grothendieck construction applied to the functor $\mathbf{Set}^{\text{op}} \to \mathbf{Cat}$ sending each set A to the category $(\mathbf{Set}/A)^{\text{op}}$. The fact that functions $f\colon A \to B$ induces an adjoint triple between $\mathbf{Set}/A$ and $\mathbf{Set}/B$, and hence between $(\mathbf{Set}/A)^{\text{op}}$ and $(\mathbf{Set}/B)^{\text{op}}$ explains the variance in (5.75) and simultaneously establishes the adjoint triple (5.74).

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 5.63, the monoidal product $\otimes$ preserves cartesian lenses; 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 lenses.

Given f and p as above, the op-cartesian lens is the lens $p \to f_!p$ obtained as the composite $p \to f^*f_!p \to f_!p$ where the first lens is the unit of the $(f_!, f^*)$ adjunction and the second is the cartesian lens 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 lenses are clearly preserved since projections in the second line match with projections in the first. □

5.7 Summary and further reading

In this chapter we discussed several of the nice properties of the category **Poly**: it has various adjunctions to **Set** and $\mathbf{Set}^{\text{op}}$, is Cartesian closed, has limits and colimits, has an epi-mono factorization system, has a vertical-cartesian factorization system, and comes with a monoidal $*$-bifibration to **Set**.

The principal monomial functor $p \mapsto p(1)y^{\Gamma p}$ discussed in Corollary 5.15 is in fact distributive monoidal, and this comes up in work on entropy [Spi22] and on noncooperative strategic games [Cap22].

5.8 Exercise solutions

Solution to Exercise 5.1.

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 3.63 #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 3.67 #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 $By^A \circ B'y^{A'} \cong B(B'y^{A'})^A \cong BB'^A y^{A'A}$ is also a monomial.

Solution to Exercise 5.6.

We complete the proof of Theorem 5.4 by exhibiting three natural isomorphisms, all special cases of (3.7), as follows.

1. By (3.7), 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 (3.7), we have the natural isomorphism

$$\begin{aligned}\mathbf{Poly}(p, A) &\cong \prod_{i \in p(1)} \sum_{a \in A} p[i]^0 \\ &\cong \prod_{i \in p(1)} \sum_{a \in A} 1\end{aligned}$$

$$\begin{aligned}&\cong \prod_{i \in p(1)} A\\&\cong \mathbf{Set}(p(1), A).\end{aligned}$$

3. By (3.7), 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 5.7.

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 5.6 #3 with $A \coloneqq 1$ to observe that

$$p(1) \cong \mathbf{Set}(1, p(1)) \cong \mathbf{Poly}(y, p).$$

Solution to Exercise 5.11.

To prove Corollary 5.10, it suffices to exhibit a natural isomorphism

$$\mathbf{Poly}(Ay^B, q) \cong \mathbf{Set}(A, q(B)).$$

Replacing p with y^B in (5.9) from Proposition 5.8, 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 5.14.

Replacing q with y in the second isomorphism in (5.9) from Proposition 5.8, we obtain the natural isomorphism

$$\mathbf{Set}(A, \mathbf{Poly}(p, y)) \cong \mathbf{Poly}(p, y^A).$$

As $\Gamma(p) = \mathbf{Poly}(p, y)$, yields the desired result.

Solution to Exercise 5.17.

We have the following chain of natural isomorphisms involving global sections:

$$\begin{aligned}\Gamma(p \otimes q) &= \mathbf{Poly}(p \otimes q, y) && (3.37)\\&\cong \mathbf{Poly}(p, [q, y]) && (4.86)\\&\cong \mathbf{Poly}(p, \Gamma(q) y^{q(1)}) && (4.82)\\&\cong \mathbf{Set}\big(p(1), \Gamma(q)\big) \times \mathbf{Set}\big(q(1), \Gamma(p)\big). && (5.16)\end{aligned}$$

Solution to Exercise 5.20.

We are given a monomorphism $f\colon 12y^{12} \to \mathbb{N}y^{\mathbb{N}}$ from Example 5.19. 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 $g_n^\sharp\colon p[g_1(n)] \to \mathbb{N}$ 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 $h_i^\sharp\colon p[g_1 i] \to 12$

which, given an input $a \in p[g_1 i]$, updates the state from i to $h_i^\sharp a = f_{g_1 i}^\sharp(g_i^\sharp a) = g_i^\sharp a \mod 12$, 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 $\mathsf{12} \subseteq \mathbb{N}$ retained, and on any input that would have caused the old system to move to a state outside of $\mathsf{12}$, the new system moves to the equivalent state (modulo 12) within $\mathsf{12}$ instead.

Solution to Exercise 5.22.

Given $p \in \mathbf{Poly}$ and a lens $f\colon p \to y$, we will use Proposition 5.21 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 (3.20) must be a monomorphism. So it follows from Proposition 5.21 that either f is an epimorphism or $p = 0$.

Solution to Exercise 5.23.

Given sets A and B, by Proposition 5.21, a lens $f\colon y^A + y^B \to y^{AB}$ 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 5.24.

Let $f\colon p \to q$ be a lens in **Poly** that is both a monomorphism and an epimorphism. We claim that f is an isomorphism. By Proposition 5.18 and Proposition 5.21, 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 5.21 says that, for each $j \in q(1)$, the induced function

$$f_j^\flat\colon q[j] \to \prod_{\substack{i \in p(1),\\ f_1 i = j}} p[i]$$

is a monomorphism. As f_1 is an isomorphism, it follows that for each $i \in p(1)$, the function

$$f_{f_1 i}^\flat\colon q[f_1 i] \to p[i]$$

is a monomorphism. But this is just the on-directions function $f_i^\sharp$ of f. From Proposition 5.18, we also know that $f_i^\sharp$ is an epimorphism. It follows that every on-directions function of f is an isomorphism. Hence f itself is an isomorphism.

Solution to Exercise 5.30.

We use (5.28) to compute various exponentials. Here $p \in \mathbf{Poly}$ and $A, B \in \mathbf{Set}$.

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 5.32.

By Theorem 5.31, 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 lens $\mathrm{eval}\colon p^q \times q \to p$ corresponding to the identity lens on p^q. The lens eval is the canonical evaluation lens.

Solution to Exercise 5.34.

1. The lens $e\colon p' \to p$ 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 lens $a\colon r \to p$ satisfying $a \fatsemi f = a \fatsemi g$, there exists a unique lens $h\colon r \to p'$ 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 5.40.

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

1. The canonical lens $\eta\colon p \to p(1)$ 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**:

$$\begin{tikzcd} p_i(1) \arrow[r, "g_1"] \arrow[d, "f_1"'] \arrow[dr, phantom, very near start, "\lrcorner"] & p(1) \arrow[d, equal] \\ 1 \arrow[r, "i"'] & p(1) \end{tikzcd}$$

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**:

$$\begin{tikzcd}
p_i[(1,i)] & p[i] \arrow[l, "g^\sharp_{(1,i)}"'] \\
0 \arrow[u, "f^\sharp_{(1,i)}"] \arrow[ur, phantom, "\lrcorner" very near end] & 0 \arrow[l, "!"] \arrow[u, "!"']
\end{tikzcd}$$

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 5.41.

1. There are many possible answers, but one lens $f\colon q \to r$, 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 $f^\sharp_1\colon 0 \to 2$ and $f^\sharp_2\colon 1 \to 1$ 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 $(f')^\sharp_1\colon 0 \to 3$ is uniquely determined, while we can let $(f')^\sharp_2\colon 1 \to 3$ pick out 3 and $(f')^\sharp_3\colon 1 \to 2$ pick out 1.
2. We compute the pullback p along with the lenses $g\colon p \to q$ and $g'\colon p \to q'$ of $q \xrightarrow{f} r \xleftarrow{f'} q'$ by following Example 5.38. 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 $f^\sharp_2$ picking out $1 \in 1 = q[2]$ and $(f')^\sharp_2$ 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 $f^\sharp_2$ still picking out $1 \in 1 = q[2]$ and $(f')^\sharp_3$ 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 5.42.

1. By Proposition 3.44, 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 $(f_1 i, f^\sharp_i \mathbin{⨾} h)$ (resp. $(g_1 i, g^\sharp_i \mathbin{⨾} h)$) 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 $f^\sharp_i \mathbin{⨾} h = g^\sharp_i \mathbin{⨾} h$.

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 $f_i^\sharp, g_i^\sharp \colon q[f_1 i] \rightrightarrows p[i]$. 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 $f_i^\sharp \mathbin{⨾} h = g_i^\sharp \mathbin{⨾} h$. 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 3.47, 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 lens $e \colon p' \to p$ constructed in the proof of Proposition 3.56, as desired.

2. By Proposition 1.37, 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 $\mathbf{Set}^{\mathbf{Set}}$. We saw in the proof of Proposition 3.56 that products in **Poly** also coincide with products in $\mathbf{Set}^{\mathbf{Set}}$. Since every limit can be computed as an equalizer of products, we can conclude that limits in **Poly** coincide with limits in $\mathbf{Set}^{\mathbf{Set}}$.

Solution to Exercise 5.44.

1. We define a lens $g \colon q \to q'$ 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 $g_v^\sharp \colon q'[c] \to q[v]$ to be the projection from the limit $q'[c]$ to its component $q[v]$.
2. To show that $s \mathbin{⨾} g = t \mathbin{⨾} g$, 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 $s_1 \mathbin{⨾} g_1 = t_1 \mathbin{⨾} g_1$. 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:

$$\begin{tikzcd}
& q[s_1(e)] \arrow[dl, "s_e^\sharp"'] & \\
p[e] & & q'[c] \arrow[ul, "g^\sharp_{s_1(e)}"'] \arrow[dl, "g^\sharp_{t_1(e)}"] \\
& q[t_1(e)] \arrow[ul, "t_e^\sharp"] &
\end{tikzcd}$$

But this is automatically true by the definition of $q'[c]$ as a limit—specifically the limit of a functor with $s_e^\sharp$ and $t_e^\sharp$ 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 $r \in$ **Poly** and lens $f \colon q \to r$ satisfying $s \mathbin{⨾} f = t \mathbin{⨾} f$, there exists a unique lens $h \colon q' \to r$ for which $f = g \mathbin{⨾} h$, 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 $f = g \mathbin{⨾} h$ to hold, we must have $f_1 = g_1 \mathbin{⨾} h_1$ on positions. But we have that $s_1 \mathbin{⨾} f_1 = t_1 \mathbin{⨾} f_1$, 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 $f_1 = g_1 \mathbin{⨾} h_1$. 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 $f = g \mathbin{⨾} h$ to hold on directions, we must have that $f_v^\sharp = h^\sharp_{g_1(v)} \mathbin{⨾} g_v^\sharp$ for each $v \in q(1)$. Put another way, given $c \in q'(1)$, we must have that $f_v^\sharp = h_c^\sharp \mathbin{⨾} g_v^\sharp$ for every $v \in V_c$. But $s \mathbin{⨾} f = t \mathbin{⨾} f$ 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 $f_v^\sharp = h_c^\sharp \mathbin{⨟} g_v^\sharp$ 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 $f_1 = g_1 \mathbin{⨟} h_1$, and that we can define h on directions such that $f_v^\sharp = h_c^\sharp \mathbin{⨟} g_v^\sharp$ for all $c \in q'(1)$ and $v \in V_c$. It follows that there exists h for which $f = g \mathbin{⨟} h$.

Solution to Exercise 5.47.

1. We characterize the lens $\epsilon \colon p(1)y \to p$ 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 lens $\eta \colon p \to y^{\Gamma(p)}$ 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 (5.48) is a pushout square is equivalent to showing that, in the diagram

$$\begin{tikzcd}
 & y \arrow[d, "\iota"] \arrow[dr, "!"] & \\
p(1)y \arrow[ur, "!"] \arrow[r, shift left, "s"] \arrow[r, shift right, "t"'] \arrow[dr, "\epsilon"'] & y + p \arrow[r, "g"] & y^{\Gamma(p)} \\
 & p \arrow[u, "\iota'"'] \arrow[ur, "\eta"'] &
\end{tikzcd} \tag{5.76}$$

in which ι, ι' are the canonical inclusions and the four triangles commute, $y^{\Gamma(p)}$ equipped with the lens g is the coequalizer of s and t. To do so, we apply Theorem 5.43 to compute the coequalizer q' of s and t. The position-set of q' is the coequalizer of $s_1 = (! \mathbin{⨟} \iota)_1$, which sends every $i \in p(1)$ to the position of $y + p$ corresponding to the summand y, and $t_1 = (\epsilon \mathbin{⨟} \iota')_1$, 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 lenses 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 (5.76) 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 5.54.

Consider the lenses $y \xrightarrow{f} y^2 + y \xrightarrow{g} y$ 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 lens $y \to y$, namely the identity, which is cartesian.

Solution to Exercise 5.56.

We wish to show that a lens $f\colon p \to q$ in **Poly** is cartesian if and only if the square on the left hand side of (5.55) 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 5.57.

The pushout of a cartesian lens is *not* necessarily cartesian. Take the pushout square (5.48). The lens $!\colon p(1)y \to y$ 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 5.61.

First, we will show that $1 \Rightarrow 3$ in Proposition 5.59. Here $f\colon p \to q$ is a cartesian lens 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 3.44 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 2.10 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 3.44. So (5.60) is a pullback, as desired.

Next, we will show that $3 \Rightarrow 1$ in Proposition 5.59, with $f\colon p \to q$ as a lens in **Poly** that is a cartesian natural transformation and $i \in p(1)$.

3. By Corollary 3.47, we have that $f_{p[i]}$ sends $(i, \mathrm{id}_{p[i]}) \mapsto (f_1 i, f_i^\sharp)$, and by Proposition 2.10, we have that $q(f_i^\sharp)$ sends $(f_1 i, \mathrm{id}_{q[f_1 i]}) \mapsto (f_1 i, f_i^\sharp)$ as well. Hence (5.62) commutes.
4. Taking $A \coloneqq q[f_1 i], B \coloneqq p[i]$, and $h \coloneqq f_i^\sharp$ in (5.60) and applying its universal property to (5.62) 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 2.10) 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 3.44) 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 5.71.

Choose $p \in$ **Poly** and $q' \in$ **Poly**$/y$. Then there is $q \in$ **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 (5.28). 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 6

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:

$$y^2 \circ (y+1) = (y+1)^2 \cong y^2 + 2y + 1.$$

In other words, $(y+1)$ is *substituted in* for the variable y in y^2. What could be simpler?

It turns out that this operation, which we will soon see is a monoidal product, has a lot to do with time. There is a strong sense—made precise in Proposition 6.2—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." This is exactly what we need to run through multiple steps of a dynamical system, the very thing we didn't know how to do in Example 4.43. We'll continue that story in Section 6.1.4.

The composition product has many surprises up its sleeve, as we'll see throughout the rest of the book.[1]

6.1 Defining the composition product

We begin with the definition of the composition product in terms of polynomials as functors.

[1]Some authors refer to $\triangleleft$ as the *substitution* product, rather than the composition product. We elected to use the composition product terminology because it provides a good noun form "the composite" for $p \triangleleft q$, whereas "the substitute" is somehow strange in English.

6.1.1 Composite functors

Definition 6.1 (Composition product)**.** Given polynomial functors p, q, we let $p \circ q$ denote their *composition product*, or 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 6.2. 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:

$$p \circ q \cong \sum_{i \in p(1)} \prod_{a \in p[i]} \sum_{j \in q(1)} \prod_{b \in q[j]} y. \tag{6.3}$$

Proof. We can rewrite p and q as

$$p \cong \sum_{i \in p(1)} y^{p[i]} \cong \sum_{i \in p(1)} \prod_{a \in p[i]} y \qquad \text{and} \qquad q \cong \sum_{j \in q(1)} y^{q[j]} \cong \sum_{j \in q(1)} \prod_{b \in q[j]} y.$$

For any set X we have $(p \circ q)(X) = p(q(X)) = p(\sum_j \prod_b X) = \sum_i \prod_a \sum_j \prod_b X$, so (6.3) is indeed the formula for the composite $p \circ q$. To see this is a polynomial, we use (1.30), which says we can rewrite the $\prod \sum$ in (6.3) as a $\sum \prod$ to obtain

$$p \circ q \cong \sum_{i \in p(1)} \sum_{\bar{j} \colon p[i] \to q(1)} y^{\sum_{a \in p[i]} q[\bar{j}(a)]} \tag{6.4}$$

(written slightly bigger for clarity), which is clearly a polynomial. □

Corollary 6.5. 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 composition product of p, i.e. n copies of p all composed with each other.[2] In particular, $p^{\triangleleft 0} = y$ and $p^{\triangleleft 1} = p$.

We repeat the important formulas from Proposition 6.2 and its proof in the new notation:

$$p \triangleleft q \cong \sum_{i \in p(1)} \prod_{a \in p[i]} \sum_{j \in q(1)} \prod_{b \in q[j]} y. \tag{6.6}$$

$$p \triangleleft q \cong \sum_{i \in p(1)} \sum_{\bar{j} \colon p[i] \to q(1)} y^{\sum_{a \in p[i]} q[\bar{j}(a)]} \tag{6.7}$$

Exercise 6.8 (Solution here). Let's consider (6.7) 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 $\overline{j_1} \colon p[1] \to q(1)$ are there?
5. For each function $\overline{j_1}$ as above, what is $\sum_{a \in p[1]} q[\overline{j_1}(a)]$?
6. How many functions $\overline{j_2} \colon p[2] \to q(1)$ are there?
7. For each function $\overline{j_2}$ as above, what is $\sum_{a \in p[2]} q[\overline{j_2}(a)]$?
8. Write out
$$\sum_{i \in p(1)} \sum_{\bar{j} \colon p[i] \to q(1)} y^{\sum_{a \in p[i]} q[\bar{j}(a)]}.$$
Does the result agree with what $p \triangleleft q$ should be? ◊

Exercise 6.9 (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. ◊

Exercise 6.10 (Solution here). Recall the closure operation $[-,-] \colon \mathbf{Poly}^{\mathrm{op}} \times \mathbf{Poly} \to \mathbf{Poly}$ for $\otimes$ from (4.75). Show that for all $A \in \mathbf{Set}$ and $q \in \mathbf{Poly}$, there is an isomorphism

$$y^A \triangleleft q \cong [Ay, q].$$

[2]When we say "the n-fold composition product of p," we mean n copies of p all composed with each other; but when we discuss an "n-fold composition product" in general, we refer to an arbitrary composition product of n polynomials that may or may not all be equal to each other. This will apply to composition products of lenses as well, once we define those.

◊

We know 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 composite $f \triangleleft g\colon p \triangleleft q \to p' \triangleleft q'$ is given by *horizontal composition*.

Definition 6.11 (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

$$p(q(X)) \xrightarrow{f_{q(X)}} p'(q(X)) \xrightarrow{p'(g_X)} p'(q'(X)) \tag{6.12}$$

obtained by composing the $q(X)$-component of f with the functor p' applied to the X-component of g.

Exercise 6.13 (Solution here). Show that we could have replaced the composite function (6.12) in Definition 6.11 with the function

$$p(q(X)) \xrightarrow{p(g_X)} p(q'(X)) \xrightarrow{f_{q'(X)}} p'(q'(X)) \tag{6.14}$$

obtained by composing p applied to the X-component of g with the $q'(X)$-component of f, without altering the definition. ◊

Remark 6.15. There are two very different notions of lens composition floating around, so we'll try to mitigate confusion by standardizing terminology here. We'll reserve the term *composite lens* for lenses $h \mathbin{⨟} j\colon r \to t$ obtained by composing a lens $h\colon r \to s$ with a lens $j\colon s \to t$, according to the composition rule of the category **Poly**. This corresponds to *vertical composition* of natural transformations. This is also the kind of composition we will mean whenever we use the verb "*compose*," if the objects of that verb are lenses.

Meanwhile, we'll use the term *composition product (of lenses)* for lenses $f \triangleleft g\colon p \triangleleft q \to p' \triangleleft q'$ obtained by applying the monoidal product functor $\triangleleft\colon \mathbf{Poly} \times \mathbf{Poly} \to \mathbf{Poly}$ on the lenses $f\colon p \to p'$ and $g\colon q \to q'$. This corresponds to *horizontal composition* of natural transformations. In this case, we'll use the verb phrase "*taking the monoidal product*."

On the other hand, we'll use the terms "composite" and "composition product" interchangeably to refer to polynomials $p \triangleleft q$, obtained by composing $p, q \in \mathbf{Poly}$ as functors or, equivalently, applying the monoidal product functor $\triangleleft$ on them—as there is no risk of confusion here.

This is another reason we tend to avoid the symbol $\circ$, preferring to use $⨟$ for vertical composition and $\triangleleft$ for horizontal composition. Of course, if you're ever confused, you can always check whether the codomain of the first lens matches up with the domain of the second. If they don't, we must be taking their monoidal product.

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 positions and directions or on corolla forests.

6.1.2 Composite positions and directions

Let us interpret our formula (6.7) for the composition product of two polynomials in terms of positions and directions. The position-set of $p \triangleleft q$ is

$$(p \triangleleft q)(1) \cong \sum_{i \in p(1)} \sum_{\bar{j} \colon p[i] \to q(1)} 1 \cong \sum_{i \in p(1)} \mathbf{Set}(p[i], q(1)). \tag{6.16}$$

In other words, specifying a position of $p \triangleleft q$ amounts to first specifying a p-position i, then specifying a function $\bar{j} \colon p[i] \to q(1)$, i.e. a q-position $\bar{j}(a)$ for each $p[i]$-direction a.

Given such a position $(i, \bar{j})$ of $p \triangleleft q$, the direction-set of $p \triangleleft q$ at $(i, \bar{j})$ is

$$(p \triangleleft q)[(i, \bar{j})] \cong \sum_{a \in p[i]} q[\bar{j}(a)]. \tag{6.17}$$

So a direction of $p \triangleleft q$ at $(i, \bar{j})$ consists of a $p[i]$-direction a and a $q[\bar{j}(a)]$-direction.

While this description completely characterizes $p \triangleleft q$, 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 the composition product of polynomials.

Back in Section 1.2, we saw how to write the instructions for choosing an element of a sum or product of sets. For instance, given a polynomial p and a set X, the instructions for choosing an element of

$$p \triangleleft X = p(X) \cong \sum_{i \in p(1)} \prod_{a \in p[i]} X$$

would be written as follows.

> To choose an element of $p(X)$:
>
> 1. choose an element $i \in p(1)$;
> 2. for each element $a \in p[i]$:
>   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. It depends on what has come before.[3] Furthermore, to highlight that these instructions are associated with some polynomial p, we will use our familiar positions and directions terminology.

> The instructions associated with a polynomial p are:

[3] In a significant sense, the composition product should be thought of as being about dependency.

1. choose a p-position i;
2. for each $p[i]$-direction a:
   2.1. choose a future.

If we think of polynomials in terms of their instructions, then (6.6) 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 a:
   2.1. choose a q-position j;
   2.2. for each $q[j]$-direction b:
      2.2.1. choose a future.

Similarly, we could write down the instructions associated with any n-fold composition product 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 (6.6) (6.7), 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 a, a q-position $\bar{j}_i(a)$;
2. for each $p[i]$-direction a and each $q[\bar{j}_i(a)]$-direction b:
   2.1. choose a future.

We will see in Section 6.1.3 that these instructions have a very natural interpretation when we depict these polynomials as corolla forests.

Exercise 6.18 (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? 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 6.19 (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 3.44 to compute the $q(X)$-component of f as a natural transformation.
2. Use Propositions 2.10 and 3.44 to compute p' applied to the X-component of g as a natural transformation.
3. Combine #1 and #2 using Definition 6.11 to compute the horizontal composite $f \triangleleft g$ of f and g as natural transformations.
4. Use Corollary 3.47 to translate the natural transformation $f \triangleleft g$ obtained in #3 to a lens $p \triangleleft q \to p' \triangleleft q'$. Verify that for each $(i, \bar{j}_i)$ in $(p \triangleleft q)(1)$ (see (6.16)), its on-positions function sends

$$(i, \bar{j}_i) \xmapsto{(f \triangleleft g)_1} \left(f_1(i), f_i^\sharp \mathbin{⨾} \bar{j}_i \mathbin{⨾} g_1\right); \tag{6.20}$$

while for each (a', b') in $(p' \triangleleft q')[(f_1(i), f_i^\sharp \mathbin{⨾} \bar{j}_i \mathbin{⨾} g_1)]$ (see (6.17)), its on-directions function sends

$$(a', b') \xmapsto{(f \triangleleft g)^\sharp_{(i, \bar{j}_i)}} \left(f_i^\sharp(a'), g^\sharp_{\bar{j}_i(f_i^\sharp(a'))}(b')\right). \tag{6.21}$$

◊

So what does Exercise 6.19 tell us about the behavior of $f \triangleleft g \colon p \triangleleft q \to p' \triangleleft q'$? By (6.20), on positions, $f \triangleleft g$ takes a p-position i and sends it to the p'-position $f_1(i)$; then for each direction a' at this position, the associated q'-position is obtained by sending a' 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 (6.20), 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 6.31.

6.1.3 Composition product on corolla forests

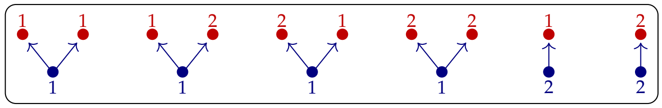
It turns out that the forest of $p \triangleleft q$ is given by grafting q-corollas onto the leaves of p-corollas in every possible way. We will demonstrate this using an example.

Let's say $p \coloneqq y^2 + y$ and $q \coloneqq y^3 + 1$, whose corolla forests we draw as follows:

(6.22)

By (6.16), choosing a position of $p \triangleleft q$ amounts to first choosing a p-root i, then choosing a q-root for every $p[i]$-leaf. So we may depict $(p \triangleleft q)(1)$ by grafting roots from the corolla forest of q to leaves in the corolla forest of p in every possible way, as follows:

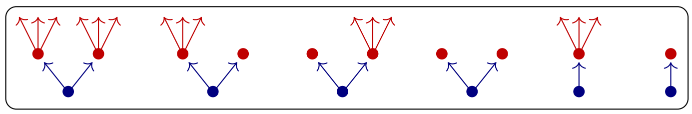
"$(p \triangleleft q)(1)$"



(6.23)

Now fix one of the positions of $p \triangleleft q$ drawn above: a p-root i and a q-root grafted to every $p[i]$-leaf. By (6.17), a direction of $p \triangleleft q$ at that position consists of a $p[i]$-leaf a and a second leaf emanating from the q-root that has been grafted on to a. In other words, in the following picture, where we have grafted not just q-roots but entire q-corollas onto leaves in p, the directions of $p \triangleleft q$ at the position corresponding to each tree are the rooted paths[4] of that tree of length 2 (we omit the labels):

"$p \triangleleft q$"

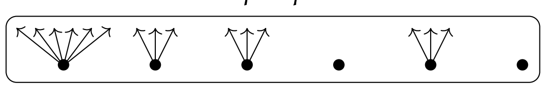


(6.24)

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 (6.24) because, to be pedantic, the corolla forest of $p \triangleleft q$ has the two levels smashed together as follows:

$p \triangleleft q$



(6.25)

Usually, we will prefer the style of (6.24) rather than the more pedantic style of (6.25).

We have now seen how to draw a single polynomial as a corolla forest, with height-1 leaves as directions; as well as how to draw a two-fold composite of polynomials as a forest of trees, with height-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 6.1.2, 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 height-n leaves as directions. You'll have an opportunity to try this in the following exercise.

Exercise 6.26 (Solution here). Use p, q as in (6.22) and $r := 2y + 1$ in the following.

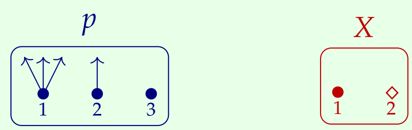
1. Draw $q \triangleleft p$.
2. Draw $p \triangleleft p$.

[4] A *rooted path* of a rooted tree is a path up the tree that starts from the root.

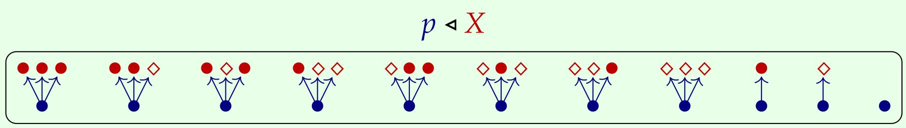
3. Draw $p \triangleleft p \triangleleft 1$.
4. Draw $r \triangleleft r$.
5. Draw $r \triangleleft r \triangleleft r$. $\Diamond$

Example 6.27 (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 $X := 2$.

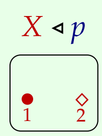


Let's see how $(y^3 + y + 1) \triangleleft 2$ looks.



It has 11 positions and no height-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 p-corolla.

Exercise 6.28 (Solution here).

1. Choose a polynomial p and draw $p \triangleleft 1$ in the style of Example 6.27.
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. $\Diamond$

In particular, this means we could write the position-set of a polynomial p interchangeably as $p(1)$ or as $p \triangleleft 1$. We'll generally write $p(1)$ when we want to emphasize the position-set as a set, and $p \triangleleft 1$ when we want to emphasize the position-set as a polynomial (albeit a constant one, with no directions).

Exercise 6.29 (Solution here). Let $\varphi\colon p \to q$ be a lens and X be a set viewed as a constant polynomial. Consider the lens $\varphi \triangleleft X\colon p \triangleleft X \to q \triangleleft X$, given by taking the composition product of φ with the identity lens on X. Show that $\varphi \triangleleft X$, when viewed as a function $p(X) \to q(X)$ between sets, is exactly the X-component of φ viewed as a natural transformation. ◊

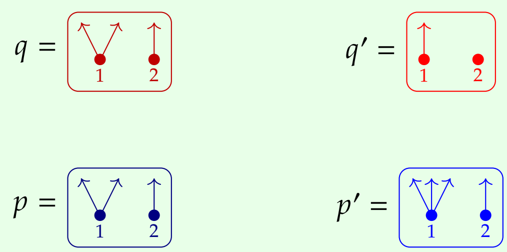
Exercise 6.30 (Solution here). For any $p \in \mathbf{Poly}$ there are natural isomorphisms $p \cong p \triangleleft y$ and $p \cong y \triangleleft p$.

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 $p \coloneqq y^3 + y + 1$. 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.

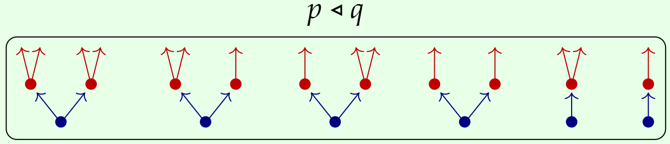
How shall we think about taking the composition product of lenses in terms of our tree pictures? We can interpret the results of Exercise 6.19 as follows.

Example 6.31. Let's take $p \coloneqq y^2 + y$, $q \coloneqq y^2 + y$, $p' \coloneqq y^3 + y$, and $q' \coloneqq y + 1$.

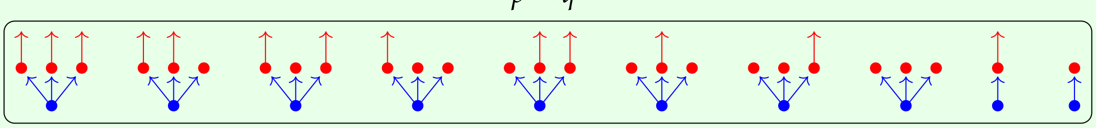


For any pair of lenses $p \to p'$ and $q \to q'$, we have a lens $p \triangleleft q \to p' \triangleleft q'$. Let's draw $p \triangleleft q$ and $p' \triangleleft q'$.

$p \triangleleft q$



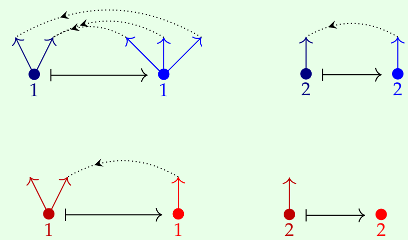
$p' \triangleleft q'$



Let's also pick a pair of lenses, $\varphi\colon p \to p'$ and $\psi\colon q \to q'$.

$\varphi\colon p \to p'$

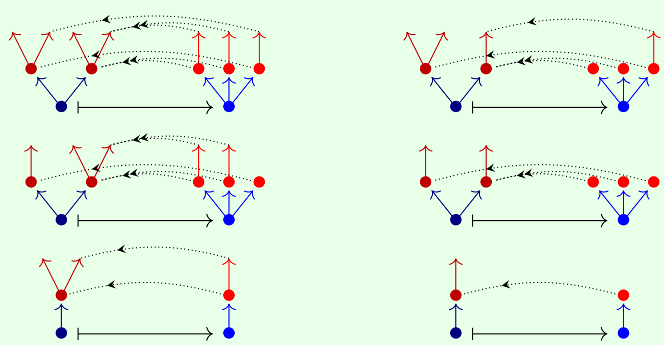
$\psi\colon q \to q'$



Then by Exercise 6.19, we can form the lens $\varphi \triangleleft \psi\colon p \triangleleft q \to p' \triangleleft q'$ as follows. On positions, we follow (6.20): for each tree t in the picture of $p \triangleleft q$, we begin by using φ_1 to send the p-corolla i that forms the bottom level of t to a p'-corolla i'. Then for each $p'[i']$-leaf a' of i', to choose which q'-corolla gets grafted onto a', we use $\varphi_i^\sharp$ to send a' back to a $p[i]$-leaf a. Since t has the corolla i as its bottom level, a is just a height-1 vertex of the tree t. So we can take the q-corolla j that is grafted onto a in t, then use ψ_1 to send j forward to a q'-corolla j'. This is the corolla we graft onto the $p'[i']$-leaf a'. All this specifies a tree t' in $p' \triangleleft q'$ that t gets sent to via $(\varphi \triangleleft \psi)_1$.

On directions, we follow (6.21): picking a direction of t' consists of picking a height-1 vertex a' and a height-2 leaf b' emanating from a'. The on-directions function $\varphi_i^\sharp$ sends a' back to a height-1 vertex a of t, and as we saw, the on-positions function ψ_1 sends the q-corolla j grafted onto a in t forward to the q'-corolla grafted onto a'. Then b' is a leaf of that q'-corolla, and $\psi_j^\sharp$ sends b' back to a leaf b emanating from a. So the on-directions function $(\varphi \triangleleft \psi)_t^\sharp$ sends the height-2 leaf b' to the height-2 leaf b.

We draw the lens $\varphi \triangleleft \psi \to p \triangleleft q \to p' \triangleleft q'$ below. To avoid clutter, we leave out the arrows for ψ_1 that show how the red corollas on the right are selected; we hope the reader can put it together for themselves.



Exercise 6.32 (Solution here). With p, q, p', q' and φ, ψ as in Example 6.31, draw the lens $\psi \triangleleft \varphi\colon q \triangleleft p \to q' \triangleleft p'$ in terms of trees as in the example. $\Diamond$

Exercise 6.33 (Solution here). Suppose p, q, and r are polynomials and you're given arbitrary lenses $\varphi\colon q \to p \triangleleft q$ and $\psi\colon q \to q \triangleleft r$. Does the following diagram necessarily commute?[5]

$$\begin{tikzcd}
q \arrow[r, "g"] \arrow[d, "\varphi"'] \arrow[dr, phantom, "?"] & q \triangleleft r \arrow[d, "\varphi \triangleleft r"] \\
p \triangleleft q \arrow[r, "p \triangleleft \psi"'] & p \triangleleft q \triangleleft r
\end{tikzcd}$$

That is, do we have $\varphi \mathbin{⨾} (p \triangleleft \psi) \overset{?}{=} \psi \mathbin{⨾} (\varphi \triangleleft r)$? ◊

6.1.4 Dynamical systems and the composition product

Back in Example 4.43, we posed the question of how to model running multiple steps of dynamical system in **Poly**. The answer lies with the composition product.

Recall that a *(dependent) dynamical system* is a lens $\varphi\colon Sy^S \to p$, where S is a set of *states* and p is a polynomial *interface*. We call Sy^S the *state system* and the on-position and on-direction functions of φ the *return* and *update* functions, respectively. More generally, we saw in Example 4.43 that we could replace the state system with a monomial $q \coloneqq Sy^{S'}$, where S' is another set, as long as there is a function $e\colon S \to S'$ (or equivalently a section $\epsilon\colon Sy^{S'} \to y$) that is bijective.

The lens models a dynamical system as follows. Every state $s \in q(1) = S$ returns a position $o \coloneqq \varphi_1(s) \in p(1)$, and every direction $i \in p[o]$ yields an updated direction $s' \coloneqq \varphi^\sharp_s(a) \in q[s] = S'$. Then to model a second step through the system, we identify the $q[s]$-direction s' with a q-position $e^{-1}(s')$, plug this position back into φ_1, and repeat the process all over again.

But this is exactly what the composition product $\varphi \triangleleft \varphi\colon q \triangleleft q \to p \triangleleft p$ does: by (6.20), its on-positions function sends the pair $(s_0, e^{-1}) \in (q \triangleleft q)(1)$, comprised of an initial state $s_0 \in q(1)$ and the function $e^{-1}\colon q[s_0] = S' \to S = q(1)$, to the pair

$$\left(\varphi_1(s_0), \varphi^\sharp_{s_0} \mathbin{⨾} e^{-1} \mathbin{⨾} \varphi_1\right) \in (p \triangleleft p)(1), \tag{6.34}$$

comprised of the initial position $o_0 \coloneqq \varphi_1(s) \in p(1)$ and a composite function

$$p[o_0] \xrightarrow{\varphi^\sharp_{s_0}} q[s_0] = S' \xrightarrow{e^{-1}} S = q(1) \xrightarrow{\varphi_1} p(1), \tag{6.35}$$

which uses the update function at s_0 and the return function to tell us what the next position o_1 will be for every possible direction i_1 we could select. Then by (6.21), the on-directions function of $\varphi \triangleleft \varphi$ sends each direction (i_1, i_2) at the position (6.34), comprised of an initial direction $i_1 \in p[o_0]$ and (setting o_1 to be the function (6.35) applied to i_1) a

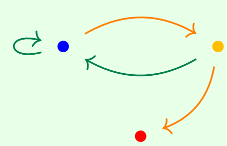
[5]When the name of an object is used in place of a morphism, we refer to the identity morphism on that object. So for instance, $\varphi \triangleleft r$ is the composition product of φ with the identity lens on r.

second direction $i_2 \in p[o_1]$, to the pair

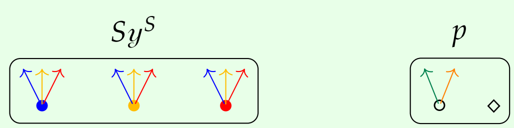
$$\left(\varphi^\sharp_{s_0}(i_1), \varphi^\sharp_{e^{-1}(\varphi^\sharp_{s_0}(i_1))}(i_2)\right),$$

comprised of directions in S' that (under e^{-1}) correspond to the next state s_1 upon selecting direction i_1 at state s_0 and the successive state s_2 upon selecting direction i_2 at s_1. In summary, at certain positions, $\varphi \triangleleft \varphi$ tells us how the dynamical system will behave when we step through it twice: starting from state s_0, returning position o_0, receiving direction i_1, updating its state to s_1, returning position o_1, receiving direction i_2, and preparing to update its state to s_2. Adding another layer, $\varphi \triangleleft \varphi \triangleleft \varphi \colon q \triangleleft q \triangleleft q \to p \triangleleft p \triangleleft p$ will tell us how the system behaves when we step through it three times; and in general, $\varphi^{\triangleleft n} \colon q^{\triangleleft n} \to p^{\triangleleft n}$ will tell us how the system behaves when we step through it n times.

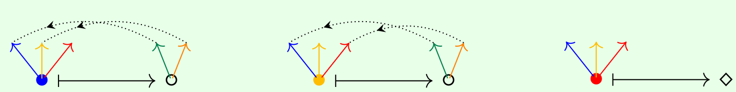
Example 6.36 (Substitution products of dynamical systems as trees). Consider the dynamical system $\varphi \colon Sy^S \to p$ with $p \coloneqq y^A + 1$, corresponding to the halting deterministic state automaton (4.24) from Exercise 4.23, depicted again here for convenience:



Below, we draw the corolla pictures for Sy^S and for p.

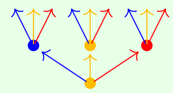


In the picture for Sy^S, the roots are the three states in $\{\bullet, \bullet, \bullet\}$ appearing in the automaton, while the leaves of each corolla correspond to the three states as well. In the picture for p, there is one corolla whose two leaves correspond to the two arrows coming out of every state—except for the halting state, which is sent to the corolla with no leaves instead. So the lens $\varphi \colon Sy^S \to p$ capturing the dynamics of the automaton can be drawn as follows:



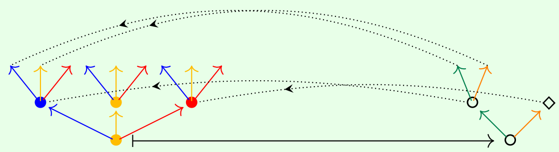
The corolla picture tells us, for example, that from the yellow state, we can go in one of two directions: the green direction, which leads us to the blue state, or the orange direction, which leads us to the red state. This describes the dynamics of the automaton one step away from the yellow state.

But what if we want to understand the dynamics of φ *two* steps away from the yellow state? Consider the following tree, corresponding to a position of the 2-fold composite $Sy^S \triangleleft Sy^S$:



(6.37)

We can follow the steps from Example 6.31 to find that the composition product of lenses $\varphi \triangleleft \varphi\colon Sy^S \triangleleft Sy^S \to p \triangleleft p$ acts on this tree as follows:



Read each tree the way you would read a decision tree, and you will find that this picture tells you exactly what the dynamics of the automaton are two steps away from the yellow state! Actually, it says that if we start from the yellow state (the root on the left, which is sent to the root on the right) and go in the orange direction (up from the root along the orange arrow to the right), the automaton will halt (as there are no more directions to follow). But if we instead go in the green direction (up from the root along the green arrow to the left), we could go in the green direction again (up the next green arrow) to arrive at a blue state (as indicated by the dashed arrow above), or instead in the orange direction to arrive at a yellow state (similarly).

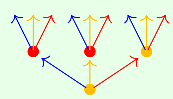
This is what we meant at the start of this chapter when we said that the substitution product has to do with time. It takes a specification φ for how a state system and an interface can interact back-and-forth—or, indeed, any interaction pattern between wrapper interfaces—and extends it to a multistep model $\varphi^{\triangleleft n}$ that simulates n successive interaction cycles over time, accounting for all possible external directions that the interface could encounter. Alternatively, we can think of $\varphi^{\triangleleft n}$ as "speeding up" the original dynamical system φ by a factor of n, as it runs n steps in one—as long as whatever's connected to its new interface $p^{\triangleleft n}$ can keep up with its pace and feed it n directions of p at a time! The lens φ tells us how the machine can run, but it is $\triangleleft$ that makes the clock tick.

Why this is not enough

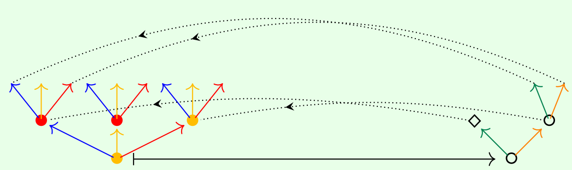
There are several pressing issues we must address, however, before we can even begin to provide a satisfying answer to everything we asked for in Example 4.43. The first is a communication issue: as you probably sensed, our set-theoretic notation for $\varphi^{\triangleleft n}$ is rather cumbersome, and that was just for $n = 2$. We could depict the behavior of our composition products of dynamical systems more clearly using tree pictures (see Example 6.36), but even that becomes infeasible in greater generality. A concise visual representation of the back-and-forth interaction of lenses would help us reason about composition products more effectively.

The second issue is more technical: to ensure that $\varphi \triangleleft \varphi$ behaves the way we want, when we specify a position of its domain $q \triangleleft q$, we have to provide not only an initial state s_0 but also the isomorphism $e^{-1} \colon S' \to S$ to let the lens know which state in S each $q[s_0]$-direction in S' should lead to. But there are many other positions (s_0, f) of $q \triangleleft q$ that we could have specified, and each $f \colon S' \to S$ associates $q[s_0]$-directions to q-positions in a different way. So $\varphi \triangleleft \varphi$ is carrying around a lot of extraneous—even misleading!—data about how our dynamical system behaves when the state system moves in the right direction but to the wrong state. Our isomorphism e^{-1} is a temporary fix, but as we pointed out in Example 4.43, it relies on the set-theoretic equality of the direction-sets of q—there's nothing inherent to the categorical structure of q in **Poly** that encodes how directions map to states, at least not yet. What are we missing?

Example 6.38 (Composition products of dynamical systems can be misleading). In Example 6.38, instead of (6.37), we could have picked the following tree, corresponding to a different (yet entirely valid) position of $Sy^S \triangleleft Sy^S$:



The composition product $\varphi \triangleleft \varphi \colon Sy^S \triangleleft Sy^S \to p \triangleleft p$ acts on this tree like so:



Now the picture tells us that, starting from the yellow state, it is the green arrow that will lead to a halting state, whereas we should somehow be able to follow the orange arrow twice!

Of course, this is nonsense—stemming from the fact that we have grafted the "wrong" corollas to each leaf when forming the position of $Sy^S \triangleleft Sy^S$ on the left. It's important to note, however, that this is nevertheless part of the data of $\varphi \triangleleft \varphi$, and we don't yet know how to tell **Poly** to rule it out.

The key to resolving both these issues lies in the next section, where we will introduce a graphical notation to help us study lenses whose codomains are composite polynomials.

6.2 Lenses to composites

Lenses to composites—that is, lenses of the form $f\colon p \to q_1 \triangleleft \cdots \triangleleft q_n$ 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, especially using polyboxes.

6.2.1 Lenses to composites as polyboxes

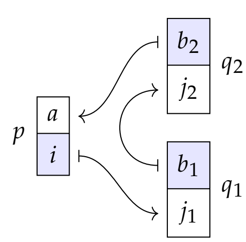
A lens $p \to q_1 \triangleleft q_2$ is an element of the set

$$
\begin{aligned}
\mathbf{Poly}(p, q_1 \triangleleft q_2) &\cong \mathbf{Poly}\left(p, \sum_{j_1 \in q_1(1)} \prod_{b_1 \in q_1[j_1]} \sum_{j_2 \in q_2(1)} \prod_{b_2 \in q_2[j_2]} y\right) && (6.6) \\
&\cong \prod_{i \in p(1)} \sum_{j_1 \in q_1(1)} \prod_{b_1 \in q_1[j_1]} \sum_{j_2 \in q_2(1)} \prod_{b_2 \in q_2[j_2]} p[i]. && (3.7)
\end{aligned}
$$

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 b_1:
>   - 1.2.1. choose a q_2-position j_2;
>   - 1.2.2. for each $q_2[j_2]$-direction b_2:
>   - 1.2.2.1. choose a $p[i]$-direction a.

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:



(6.39)

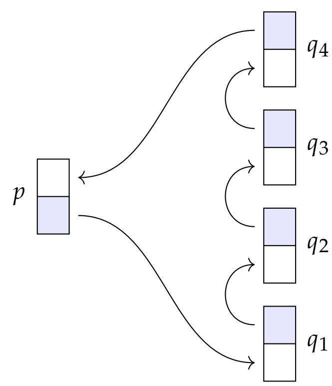
Whenever we draw two pairs of polyboxes on top of each other, as we do with the polyboxes for q_1 and q_2 above on the right, we are indicating that the entire column of polyboxes depicts the composite of the polynomials depicted by each individual pair. So the column of polyboxes on the right represents the composite $q_1 \triangleleft q_2$. In particular, the position in the lower box of the top pair is the position associated with the direction in the upper 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 6.40.

In fact, (3.34), (3.13), and (6.39) are the respective polybox depictions of the $n = 0, n = 1$, and $n = 2$ cases of lenses $p \to q_1 \triangleleft \cdots \triangleleft q_n$ to n-fold composites (we consider the monoidal unit y of $\triangleleft$ to be the 0-fold composite, and a 1-fold composite is just a polynomial on its own). In general, for any $n \in \mathbb{N}$, we can apply

$$
\begin{aligned}
\mathbf{Poly}(p, q_1 \triangleleft \cdots \triangleleft q_n) &\cong \mathbf{Poly}\left(p, \sum_{j_1 \in q_1(1)} \prod_{b_1 \in q_1[j_1]} \cdots \sum_{j_n \in q_n(1)} \prod_{b_n \in q_n[j_n]} y\right) && (6.6)\\
&\cong \prod_{i \in p(1)} \sum_{j_1 \in q_1(1)} \prod_{b_1 \in q_1[j_1]} \cdots \sum_{j_n \in q_n(1)} \prod_{b_n \in q_n[j_n]} p[i], && (3.7)
\end{aligned}
$$

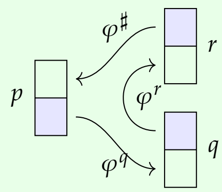
so the polybox depiction of $p \to q_1 \triangleleft \cdots \triangleleft q_n$ generalizes analogously. For example, here are the polyboxes corresponding to a lens to a 4-fold composite:



These lenses to n-fold composites lend themselves to a very natural interpretation in terms of our decision-making language. Each of p's menus is passed forward to a menu for q_1 to choose from. For every option that q_1 may choose, there is then also a menu for q_2 to choose from. Then for every option that q_2 may choose, there is a menu for q_3 to choose from, and so on, all the way until q_n has chosen an option. Together, all the options that $q_1, \ldots, q_n$ chose then inform the option that p should select from its original menu.

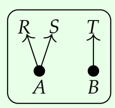
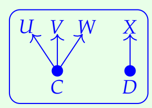
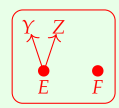
A lens $p \to q_1 \triangleleft \cdots \triangleleft q_n$ is a multi-step policy for p to make decisions by asking for decisions from q_1, then q_2, etc., all the way to q_n, then interpreting the results.

Example 6.40 (Lenses $p \to q \triangleleft r$). Consider a lens $\varphi\colon p \to q \triangleleft r$. Let's label the three arrows in the lens's polybox depiction:

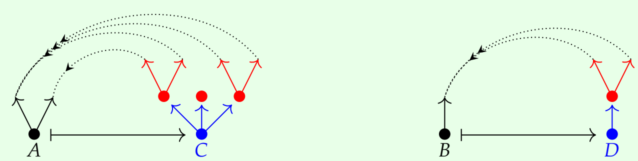


So the on-position function of φ can be split into two parts: a function $\varphi^q \colon p(1) \to q(1)$ and, for each $i \in p(1)$, a function $\varphi^r_i \colon q[\varphi^q(i)] \to r(1)$. Then the on-directions function $\varphi^\sharp_i \colon (q \triangleleft r)[\varphi_1(i)] \to p[i]$ takes the 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 $r \coloneqq \{E\}y^{\{Y,Z\}} + \{F\}$.

$p =$  $q =$  $r =$ 

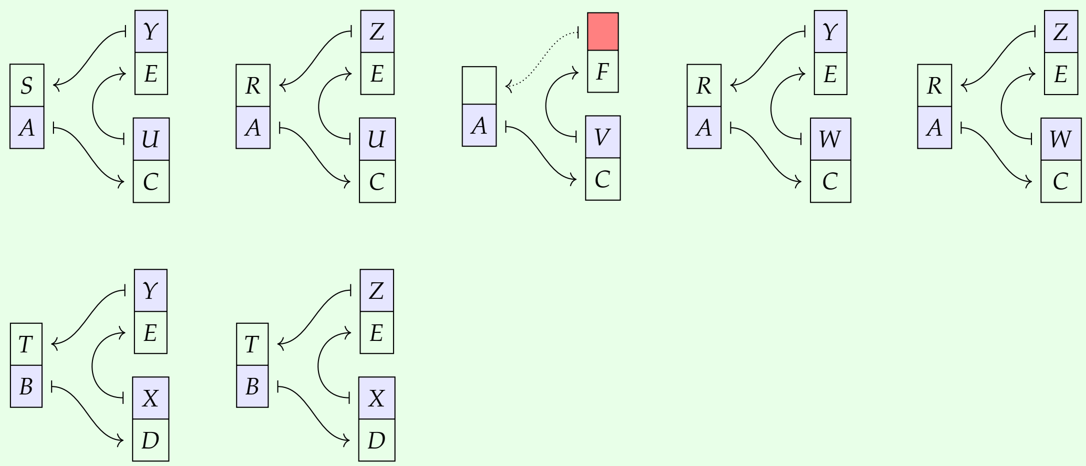
Here is a tree picture of a lens $\varphi \colon p \to q \triangleleft r$:



If we write φ as the corresponding triple $(\varphi^q, \varphi^r, \varphi^\sharp)$, then we have

$$
\begin{gathered}
\varphi^q(A) = C, \quad \varphi^q(B) = D; \\
\varphi^r_A(U) = E, \quad \varphi^r_A(V) = F, \quad \varphi^r_A(W) = E; \\
\varphi^r_B(X) = E; \\
\varphi^\sharp_A(U,Y) = S, \quad \varphi^\sharp_A(U,Z) = R, \quad \varphi^\sharp_A(W,Y) = R, \quad \varphi^\sharp_A(W,Z) = R; \\
\varphi^\sharp_B(X,Y) = T, \quad \varphi^\sharp_B(X,Z) = T.
\end{gathered}
$$

Polyboxes display the same data in a different format:

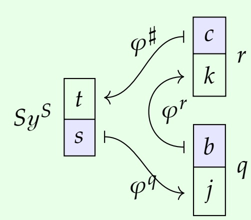


As before, keep in mind that each arrow of a lens depends not only on the box it emerges from, but also on every box that came before it in our usual reading order (lower left to lower right to upper right to upper left).

The third set of polyboxes, where the left blue box has been filled with an A and the lower right blue box has been filled with a V, is worth highlighting: as $\varphi^r_A(V) = F$, but $r[F] = \varnothing$, it is impossible to write a direction of r at F to go in the upper right box. To indicate this, we color the upper right box red and leave the arrow emerging from it dashsed.

Example 6.41 (Dynamical systems with composite interfaces). We explored dynamical systems with product interfaces in Section 4.3.1 and parallel product interfaces in Section 4.3.2. How about dynamical systems with composite interfaces? We now have all the tools we need to characterize them.

By the previous example, a dynamical system $\varphi\colon Sy^S \to q \triangleleft r$ can be drawn as



We can interpret the behavior of this system as follows. Rather than a single interface $q \triangleleft r$, we view φ as having two interfaces that must be interacted with in succession, q followed by r.

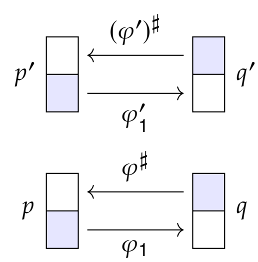
Given the current state $s \in S$, the system feeds it into φ^q to return a position j of the first interface q. Upon receiving a direction $b \in q[j]$, it then uses φ^r to return another position k (dependent on the state s and the direction b), this time belonging to the second interface r. Once a second direction c is received, this time from $r[k]$, the system updates its state by feeding the current state s and the pair of directions (b, c) it received into $\varphi^\sharp$, yielding a new state $t \in S$.

That's a lot of words, which is why the polybox picture is so helpful: by following the arrows, we can see that a dynamical system with a composite interface actually captures a very natural type of interaction! Mixing our metaphors a little, φ could model a system that displays cascading menus, where selecting an option b on the first menu j opens up a second menu k. It is only when the interacting agent selects an option c from this second menu that both choices are sent back to the state system, which updates its state accordingly.

All this generalizes to n-fold composite interfaces exactly how you'd expect: a dynamical system with interface $q_1 \triangleleft \cdots \triangleleft q_n$ returns a position and receives a direction through interface q_1, then accordingly returns a position and receives a direction through interface q_2, and so on, until it returns a position (according to the current state and all previous directions) and receives a direction through q_n, whereupon it updates its state according to the n directions it received along with the current state.

6.2.2 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 using polyboxes by following the instructions from (6.20) and (6.21), we would really just be stacking the polyboxes for the constituent lenses on top of each other. For example, given lenses $\varphi\colon p \to q$ and $\varphi'\colon p' \to q'$, here is $\varphi \triangleleft \varphi'$ drawn as polyboxes:

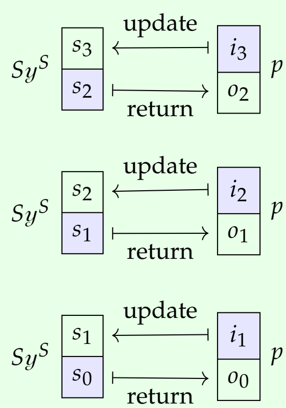


(6.42)

What differentiates this from simply writing down the polyboxes for φ and the polyboxes for φ' is that we are explictly associating the position that will fill the lower box of p' with the direction that will fill the upper box of p, and likewise the position that will fill the lower box of q' with the direction that will fill the upper box of q. Moreover, we have the user fill out the lower set of boxes first and work their way up, so that, in particular, they can use the information they obtained from the behavior of φ_1 and $\varphi^\sharp$ to decide what to put in the lower box of p'. So this really does depict a lens $p \triangleleft p' \to q \triangleleft q'$.

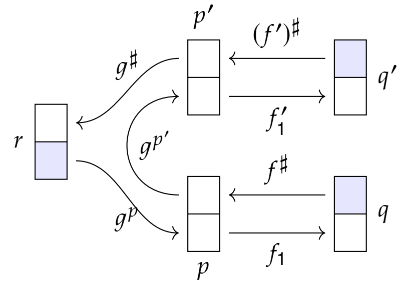
How does (6.42) relate to our usual polybox depiction of a lens to a composite, as in (6.39), but with the domain also replaced with a composite? A user who interacts with (6.42) can fill the lower set of polyboxes (the ones for φ) first, ignoring the upper set of polyboxes (the ones for φ') until the entire lower half is filled. Alternatively, after they fill in the lower box of p, but before they fill in anything else, they can already decide what position to put in the lower box of p' for every possible direction that could end up in the upper box of p. By (6.16), such a choice is equivalent to picking a position of the composite $p \triangleleft p'$. Then by (6.20), following just the bottom arrow φ_1 leads to the corresponding position of q given by $\varphi \triangleleft \varphi'$, while filling in the upper box of q and following $\varphi^\sharp$, then φ'_1 leads to the position of q' that goes in the bottom box of q'. Finally, once the user fills in the upper box of q', following the top arrow $(\varphi')^\sharp$ completes the specification of a direction of $p \triangleleft p'$. In this way, (6.42) can be thought of as a special case of (6.39).

Example 6.43 (Dynamical systems and the composition product, revisited). In Section 6.1.4, we explained how the n-fold composition product $\varphi^{\triangleleft n}$ of a dynamical system $\varphi\colon Sy^S \to p$ models the behavior of running through the system n times, provided we choose the positions of $(Sy^S)^{\triangleleft n}$ appropriately. We can visualize this behavior using polyboxes—for example, here's what the $n = 3$ case looks like:



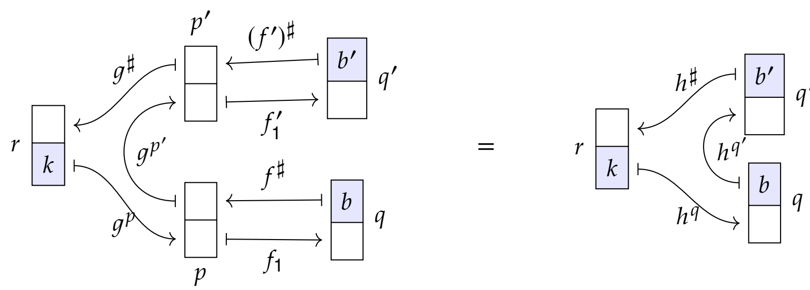
It is now patently obvious what $\varphi^{\triangleleft 3}$ does from this picture, as long as we know how to read polyboxes (and we could probably make a pretty good guess even if we didn't!). This resolves the first issue we raised in Section 6.1.4, page 192: we now have a concise way of depicting the n-fold composite of a dynamical system. The second issue becomes clear when we look at which boxes are blue along the left: we would really like the position s_1 to be entered above the direction s_1 automatically, the s_2 entered above s_2 automatically, etc. rather than having to specify the contents of those blue boxes manually. We shouldn't even having the option to fill those blue boxes in with anything else. We'll see how to address this issue shortly in Example 6.44.

We make a big deal out of it, but (6.42) really is just the polyboxes of two separate lenses drawn together. Where such polyboxes truly get interesting is when we compose them with polyboxes that look like (6.39). That is, given a lens $g\colon r \to p \triangleleft p'$, consider the polyboxes for $g \fatsemi (f \triangleleft f')$:



There's a lot going on with this lens! To fill out these polyboxes, we start from the lower box of r, go all the way right to the lower box of q, loop back left, up, and right again to the lower box of q', then travel left all the way back to the upper box of r.

Say we knew that $g \fatsemi (f \triangleleft f')$ were equal to some other lens $h\colon r \to q \triangleleft q'$:

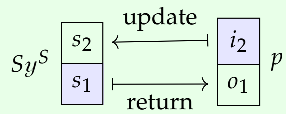


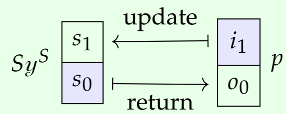
We've filled in the corresponding blue boxes on either side of the equation with the same entries. So if these two sets of polyboxes really do depict the same lens, each of the three white boxes in the domain and codomain on the left should end up with the same entry as the corresponding white box on the right (although the intermediary mechanics may differ). Then if we follow the arrows in order on either side, matching up the white boxes in the domain and codomain along the way, we can read off three equations:

$$g^p \fatsemi f_1 = h^q, \qquad f^\sharp_{g^p(k)} \fatsemi g^{p'}_k \fatsemi f'_1 = h^{q'}_k, \qquad \text{and} \qquad (f')^\sharp_{g^{p'}\left(f^\sharp_{g^p(k)}(b)\right)} \fatsemi g^\sharp_k = h^\sharp_k.$$

The converse holds as well: if the three equations above all hold, then $g \fatsemi (f \triangleleft f') = h$. We will read equations off of polyboxes like this repeatedly in the rest of the book.

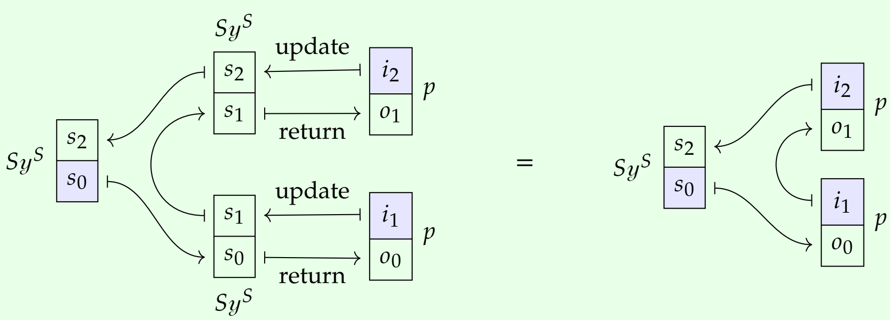
Example 6.44 (Transition lenses for state systems). In Example 6.43, we saw that the 2-fold composition product $\varphi^{\triangleleft 2}$ of a dynamical system $\varphi\colon Sy^S \to p$ can be drawn as follows:



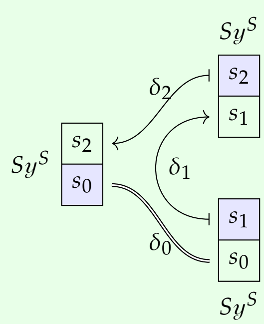


This *almost* models the behavior of running through the system twice, except that we should really only have one blue box on the domain side—the one we fill with the initial state s_0. The second blue box on the domain side, the one we fill with s_1, should instead be filled automatically with the same state as the direction s_1 in the white box below it.

In fact, it would be nice if the domain were still just Sy^S. Then we would have a lens $Sy^S \to p \triangleleft p$ that takes an initial state $s_0 \in S$ and runs the original system φ twice, returning two positions and receiving two directions before stopping at the new state s_2. But we just learned how to take a composition product of lenses such as $\varphi^{\triangleleft 2}\colon Sy^S \triangleleft Sy^S \to p \triangleleft p$ and convert its domain to a new polynomial, say Sy^S, with only one blue box on the domain side—just compose it with another lens $Sy^S \to Sy^S \triangleleft Sy^S$ like so:



We'll denote our new lens $Sy^S \to Sy^S \triangleleft Sy^S$ on the far left by δ. Let's take a closer look at how δ behaves on its own, labeling its arrows for ease of reference:



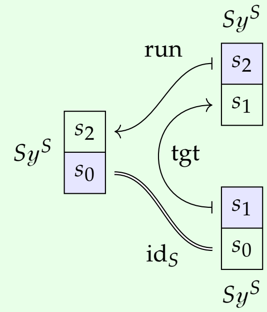
We can see from this picture that δ arises very naturally from the structure of Sy^S; indeed, every state system can be equipped with such a lens, just as every state system can be equipped with a do-nothing section from Example 4.43. The bottom arrow $\delta_0 \colon S \to S$ sending $s_0 \mapsto s_0$ is the identity on the position-set S: it sends each state to itself. We use a double arrow to denote this equality. While the middle arrow δ_1 also looks like an identity arrow, remember that we should think of it as depending on the left blue box as well; so it is really a function $\delta_1 \colon S \times S \to S$ sending (s_0, s_1), a position-state s_0 and a direction at s_0 corresponding to state s_1, to the new position-state s_1. Similarly, $\delta_2 \colon S \times S \times S \to S$ sends (s_0, s_1, s_2), where the last coordinate is the direction at position-state s_1 corresponding to state s_2, to the direction at position-state s_0 that also corresponds to state s_2.

Notice the crucial role that δ_1 plays here: it matches up every direction at a given state to the new state that the direction in question should point to, encoding how the state system transitions from one state to the next. We have been labeling each direction by the state that it points to, but we should really think of all the directions of Sy^S as pairs of position-states $(s, t) \in S \times S$, where (s, t) is a direction at s that represents the transition from state s to state t. The structure of the polynomial Sy^S tells us the state that each direction transitions from, but it is δ_1 that tells us the new state $\delta_1(s, t) = t$ that (s, t) transitions to. For that reason, we call δ the *transition lens* of the state system Sy^S, and we call $\delta_1 \colon S \times S \to S$ the *target function*, relabeling it tgt for short, indicating where each direction leads.

This is exactly what we wanted back in Example 4.43: a way to encode which directions of a state system point to which positions in the language of **Poly**. Expressions in that language are lenses such as δ, and polyboxes like the ones above are the way we write them down.

We highlighted δ_1 above, but the arrows δ_0 and δ_2 are no less important. As an identity function, $\delta_0 = \mathrm{id}_S$ remembers the initial state s_0 as we shift from its original setting Sy^S, where we can only move one state away from s_0, to a new setting $Sy^S \triangleleft Sy^S$, where we can think about all the ways to move two states away from s_0. Meanwhile, δ_2 shifts us from the two-state setting back to the one-state setting while ensuring a sort of transitive coherence condition: the state where we end up after moving from s_0 through s_1 to s_2 is the same state we would end up at if we had moved from s_0 directly to s_2. We will call it the *run function*, because it runs a sequence of two transitions together, and because it is what keeps the system running as it tells the original state system to actually move along one of its directions.

Here is another picture of the transition lens δ of Sy^S with our new arrow names:



This resolves the second issue we raised in Section 6.1.4, page 192 for the case of $n = 2$, giving us a dynamical system $\delta \mathbin{⨾} (\varphi \triangleleft \varphi)\colon Sy^S \to p \triangleleft p$ that simulates stepping through the system φ twice. But what about more general values of n?

We already have a way of talking about the $n = 0$ case: that is what the do-nothing section $\epsilon\colon Sy^S \to y$ models. But there is some overlap in how ϵ matches up state-positions with directions and how δ does. Put another way, if ϵ tells us how to do nothing, and δ tells us how to do two things, then we had better check that if one of the two things we do is nothing, then that's the same as doing just one thing. Can we ensure that ϵ and δ agree on what they are saying about our state system?

Then for $n = 3$, we would like a dynamical system $Sy^S \to p \triangleleft p \triangleleft p$ that simulates stepping through φ three times. One way to do this would be to compose $\varphi^{\triangleleft 3}$ with a lens of the form $Sy^S \to \left(Sy^S\right)^{\triangleleft 3}$ that we obtain by extending δ from modeling two-step transitions to three-step transitions. But there are two ways to derive a lens with codomain $\left(Sy^S\right)^{\triangleleft 3}$ from δ: we could either take $\delta \mathbin{⨾} \left(\delta \triangleleft \mathrm{id}_{Sy^S}\right)$, or we could take $\delta \mathbin{⨾} \left(\mathrm{id}_{Sy^S} \triangleleft \delta\right)$ For larger values of n, there are even more possibilities for what we could do. But there should really only be one dynamical system that models stepping through φ a fixed number of times. How do we guarantee that all these different ways of extending δ to n-step transitions end up telling us the same thing?

In summary, we need some kind of compatibility condition between ϵ and δ, as well as some kind of associativity condition on δ to guarantee that it can be extended coherently. In fact, we already have all the tools we need to characterize these conditions: we'll see exactly how to state the properties we want in the next chapter. And if this is all starting to sound suspiciously familiar, you're not wrong—but we'll save that surprise for the next chapter as well.

Exercise 6.45 (Solution here). Let $S := \mathbb{N}$ and $p := \mathbb{R}y^1$, and define $\varphi\colon Sy^S \to p$ to be the dynamical system with return function $\varphi_1(k) := k$ and update function $\varphi^\sharp_k(1) := k + 1$.

1. Draw the polyboxes for φ and describe its dynamics: what does 1 run through the system look like?

2. Let $\delta \colon Sy^S \to Sy^S \triangleleft Sy^S$ be the transition lens of Sy^S, and draw the polyboxes for the new system $\delta \mathbin{⨟} (\varphi \triangleleft \varphi) \colon Sy^S \to p \triangleleft p$. Describe its dynamics: how does it model 2 runs through the system? ◊

Exercise 6.46 (Solution here). As a lens whose domain is a state system, the transition lens $\delta \colon Sy^S \to Sy^S \triangleleft Sy^S$ of a state system Sy^S can be interpreted as a standalone dynamical system. Describe the dynamics of this system. ◊

6.3 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. We'll focus on how $\triangleleft$ interacts with other constructions on **Poly** that we introduced in previous chapters.

6.3.1 Interaction with products and coproducts

It turns out that the composition product behaves well—albeit asymmetrically—with products and coproducts.

Proposition 6.47 (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:

$$(p + q) \triangleleft r \cong (p \triangleleft r) + (q \triangleleft r) \tag{6.48}$$

and

$$pq \triangleleft r \cong (p \triangleleft r)(q \triangleleft r). \tag{6.49}$$

More generally, given a set A and polynomials $(q_a)_{a \in A}$, we have the following natural isomorphisms:

$$\left(\sum_{a \in A} q_a\right) \triangleleft r \cong \sum_{a \in A} (q_a \triangleleft r) \tag{6.50}$$

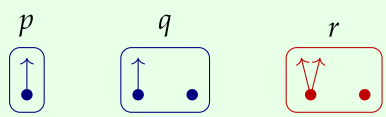
and

$$\left(\prod_{a \in A} q_a\right) \triangleleft r \cong \prod_{a \in A} (q_a \triangleleft r) \tag{6.51}$$

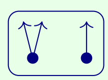
Proof. Formally, this comes down to the fact that (co)products of functors $\mathbf{Set} \to \mathbf{Set}$ are computed pointwise (Proposition 1.37) and that (co)products in **Poly** coincide with (co)products in $\mathbf{Set}^{\mathbf{Set}}$ (Propositions 3.3 and 3.56). One could instead give an explicit proof using (6.6); this is done in Exercise 6.52. In fact, we will see yet another proof of (6.51) (and thus (6.49)) in Exercise 6.70 #2. □

Exercise 6.52 (Solution here). Prove Proposition 6.47 using the explicit formula for $\triangleleft$ given in (6.6) by manipulating sums and products. ◊

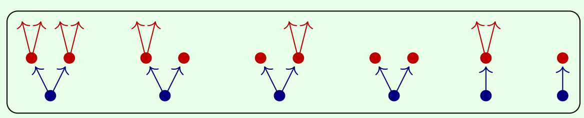
Example 6.53 (Picturing the left distributivity of $\triangleleft$ over $\times$). We want an intuitive understanding of the left distributivity given by (6.49). Let $p := y$, $q := y + 1$, and $r := y^2 + 1$, as shown here:



Then $pq \cong y^2 + y$ can be drawn as follows, with each corolla comprised of a p-corolla and a q-corolla with their roots glued together:

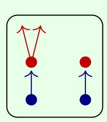
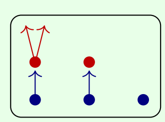
$pq \cong$ 

We can therefore draw $pq \triangleleft r$ by grafting r-corollas to leaves of pq in every way, as follows:

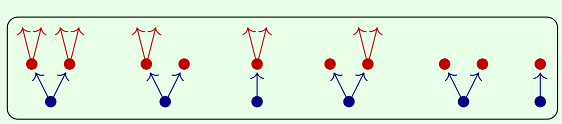
$pq \triangleleft r \cong$ 

So each tree in $pq \triangleleft r$ is obtained by grafting together the roots of a p-corolla and a q-corolla, then attaching r-corollas to each leaf.

Alternatively, we can compute $p \triangleleft r$ and $q \triangleleft r$ seperately, grafting r-corollas 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 grafting r-corollas to each leaf of a p-corolla and a q-corolla before gluing their roots together.

But it doesn't matter if we graft r-corollas onto leaves first, or if we glue the roots of p- and q-corollas together first–the processes are equivalent. Hence the isomorphism $pq \triangleleft r \cong (p \triangleleft r)(q \triangleleft r)$ holds.

Exercise 6.54 (Solution here). Follow Example 6.53 with coproducts $(+)$ in place of products $(\times)$: use pictures to give an intuitive understanding of the left distributivity given by (6.48). $\Diamond$

Exercise 6.55 (Solution here). Show that for any set A and polynomials p, q, we have an isomorphism $A(p \triangleleft q) \cong (Ap) \triangleleft q$. $\Diamond$

In Section 6.3.2, 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 6.56 (Solution here). Show that the distributivities of Proposition 6.47 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 $p \triangleleft (q + r) \not\cong (p \triangleleft q) + (p \triangleleft r)$. $\Diamond$

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.

6.3.2 Interaction with limits on the left

We saw in Theorem 5.33 that **Poly** has all limits, and we saw in Exercise 5.42 that these limits coincide with limits in $\mathbf{Set}^{\mathbf{Set}}$. Hence the argument in the proof of Proposition 6.47 by appealing to Proposition 1.37 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 6.57 (Meyers)**.** The composition product is left coclosed. That is, there exists a left coclosure operation, which we denote $\begin{bmatrix} - \\ - \end{bmatrix} \colon \mathbf{Poly}^{\mathrm{op}} \times \mathbf{Poly} \to \mathbf{Poly}$, such that there is a natural isomorphism

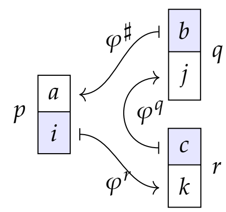
$$\mathbf{Poly}(p, r \triangleleft q) \cong \mathbf{Poly}\left(\begin{bmatrix} q \\ p \end{bmatrix}, r\right). \tag{6.58}$$

In particular, the left coclosure operation sends $q, p \in \mathbf{Poly}$ to

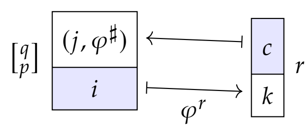
$$\begin{bmatrix} q \\ p \end{bmatrix} := \sum_{i \in p(1)} y^{q(p[i])}. \tag{6.59}$$

Proof. We present an argument using polyboxes; we leave it to the reader to write this proof in more standard mathematical notation in Exercise 6.60.

As in Example 6.40, a lens $\varphi\colon p \to r \triangleleft q$ can be written as follows:



But this is equivalent to the following gadget (to visualize this equivalence, imagine leaving the positions box for p, the arrow φ^r, and the polyboxes for r untouched, while dragging the polyboxes for q leftward to the directions box for p, merging all the data from q and the arrows φ^q and $\varphi^\sharp$ into a single on-directions arrow and directions box):



Here the on-directions function encodes the behaviors of both φ^q and $\varphi^\sharp$ by sending each $r[k]$-direction c to both a q-position j, as φ^q does, and a function sending $q[j]$-directions to $p[i]$-directions, as $\varphi^\sharp$ does. So the polyboxes on the left represent a polynomial whose positions are the same as those of p, but whose directions at $i \in p(1)$ are pairs $(j, \varphi^\sharp)$ consisting of a q-position j and a function $\varphi^\sharp\colon q[j] \to p[i]$. Such pairs are precisely the elements of $q(p[i])$, so the polynomial represented by the polyboxes on the left is indeed the one defined as $\left[\begin{smallmatrix} q \\ p \end{smallmatrix}\right]$ in (6.59). It follows that there is a natural isomorphism between lenses $p \to r \triangleleft q$ and lenses $\left[\begin{smallmatrix} q \\ p \end{smallmatrix}\right] \to r$. $\square$

Exercise 6.60 (Solution here). Translate the polyboxes proof of Proposition 6.57 into standard mathematical notation, i.e. the $\sum$ and $\prod$ notation we have been using up till now. $\Diamond$

Remark 6.61. The proof you came up with in Exercise 6.60 may be more obviously rigorous and concise than the one we presented in the main text. But polyboxes help us see right on paper exactly what is going on in this adjunction: how data on the codomain-side of a lens $p \to r \triangleleft q$ can be simply repackaged and transferred to the domain-side of a new lens $\left[\begin{smallmatrix} q \\ p \end{smallmatrix}\right] \to r$.

Exercise 6.62 (Solution here). In stating Proposition 6.57, we implicitly assumed that $\left[\begin{smallmatrix} - \\ - \end{smallmatrix}\right]$ is a functor $\mathbf{Poly}^{\mathrm{op}} \times \mathbf{Poly} \to \mathbf{Poly}$. Here we show that this is indeed the case.

1. Given a polynomial q and a lens $\varphi\colon p \to p'$, to what lens $\left[\begin{smallmatrix} q \\ p \end{smallmatrix}\right] \to \left[\begin{smallmatrix} q \\ p' \end{smallmatrix}\right]$ should the covariant functor $\left[\begin{smallmatrix} q \\ - \end{smallmatrix}\right]$ send φ? Prove that your construction is functorial.

2. Given a polynomial p and a lens $\psi\colon q' \to q$, to what lens $\begin{bmatrix} q \\ p \end{bmatrix} \to \begin{bmatrix} q' \\ p \end{bmatrix}$ should the contravariant functor $\begin{bmatrix} - \\ p \end{bmatrix}$ send ψ? Prove that your construction is functorial. ◊

Exercise 6.63 (Solution here). In personal communication, Todd Trimble noted (the in-retrospect-obvious fact) that the left coclosure can be thought of as a left Kan extension

$$\begin{tikzcd}
\mathbf{Set} \arrow[r, "p"] \arrow[d, "q"'] \arrow[dr, phantom, "\Downarrow"] & \mathbf{Set} \\
\mathbf{Set} \arrow[ur, "{\begin{bmatrix} q \\ p \end{bmatrix}}"'] &
\end{tikzcd}$$

Verify this. ◊

Exercise 6.64 (Solution here). Let A and B be sets, and let p and q be polynomials.

1. Prove that the following natural isomorphism holds:
$$\mathbf{Poly}(Ay^B, p) \cong \mathbf{Set}(A, p(B)). \tag{6.65}$$
2. Prove that the following natural isomorphism holds:
$$\mathbf{Poly}\left(Ay \triangleleft p \triangleleft y^B, q\right) \cong \mathbf{Poly}\left(p, y^A \triangleleft q \triangleleft By\right). \tag{6.66}$$
(Hint: Break the isomorphism down into two parts. You may find (5.9) helpful.)

◊

Example 6.67 (Dynamical systems as coalgebras). Taking $A = B = S \in \mathbf{Set}$ in (6.65), we find that there is a natural isomorphism between dynamical systems $Sy^S \to p$ and functions $S \to p(S)$. Such a function is known as a *coalgebra for the functor* p or a *p-coalgebra*.[6]

Coalgebras as models of dynamical systems have been studied extensively in the context of computer science, most notably by Jacobs in [Jac17]. Indeed, much of what we developed in stems from the theory of coalgebras. The coalgebraic perspective has the benefit of staying in the familiar category of sets; moreover, it can be generalized to functors $\mathbf{Set} \to \mathbf{Set}$ that are not polynomial, although many of the interesting examples are.

On the other hand, we have already seen that viewing dynamical systems as lenses $Sy^S \to p$ rather than as functions $S \to p(S)$ has the benefit of isolating the internal state system to the domain and the external interface to the codomain, aiding both intuition and functionality. Plus, our adjunction lets us switch to the coalgebraic perspective whenever we see fit: $\mathbf{Poly}$ lets us talk about both.

Proposition 6.68 (Left preservation of limits)**.** The operation $\triangleleft$ preserves limits on the left (up to natural isomorphism). That is, if $\mathcal{J}$ is a category, $p_-\colon \mathcal{J} \to \mathbf{Poly}$ is a functor, and $q \in \mathbf{Poly}$ is a polynomial, then there is a natural isomorphism

$$\left(\lim_{j \in \mathcal{J}} p_j\right) \triangleleft q \cong \lim_{j \in \mathcal{J}} (p_j \triangleleft q). \tag{6.69}$$

Proof. By Proposition 6.57, the functor $(- \triangleleft q)\colon \mathbf{Poly} \to \mathbf{Poly}$ is the right adjoint of the functor $\left[\begin{smallmatrix} q \\ - \end{smallmatrix}\right]\colon \mathbf{Poly} \to \mathbf{Poly}$, and right adjoints preserve limits. □

Exercise 6.70 (Solution here).

1. Complete Exercise 6.28 #3 using (6.69) and (6.50).
2. Deduce (6.51) using (6.69). ◊

6.3.3 Interaction with limits on the right

So $\triangleleft$ preserves limits on the left. How about limits on the right? We saw in Exercise 6.56 that $\triangleleft$ does not even preserve products on the right, so it certainly does not preserve all limits. But it turns out that there is a special class of limits that $\triangleleft$ does preserve on the right.

Definition 6.71 (Connected limit)**.** A *connected limit* is one whose indexing category $\mathcal{J}$ is nonempty and connected. That is, $\mathcal{J}$ has at least one object, and any two objects are connected by a finite zigzag of arrows.

Example 6.72. 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.

[6] There are two versions of coalgebras we are interested in (and more that we are not) with distinct definitions: a *coalgebra for a functor*, which is the version used here, and a *coalgebra for a comonad*, which is a coalgebra for a functor with extra conditions that we will introduce later in Section 7.3.3. The version we are using will usually be clear from context—here, for example, we do not expect p to be a comonad—but we will try to be explicit with our terminology whenever the interpretation may be ambiguous.

Connected limits are intimately related to slice categories, which we defined back in Definition 5.66. For example, products in a slice category $\mathcal{C}/c$ are just pullbacks in $\mathcal{C}$, allowing us to view a non-connected limit as a connected one. By relating **Poly** to its slice categories via an adjunction, we'll be able to show that $\triangleleft$ preserves connected limits. (An alternative proof of this fact can be found in [GK12, Proposition 1.16].)

Recall that objects in a slice category $\mathcal{C}/c$ are just morphisms with codomain c. For ease of notation, we'll often suppress the actual morphism and just write down the name of its domain when there is a canonical choice for the morphism, or when it is clear from context. So for example, on the left hand side of (6.74) below, p represents the lens $f\colon p \to q \triangleleft 1$ and $q \triangleleft r$ represents the lens $q \triangleleft !\colon q \triangleleft r \to q \triangleleft 1$, both objects in the slice category $\mathbf{Poly}/q \triangleleft 1$.

Proposition 6.73. Given polynomials $p, q, r \in \mathbf{Poly}$ and a function $f\colon p \to q \triangleleft 1$, there is a natural isomorphism

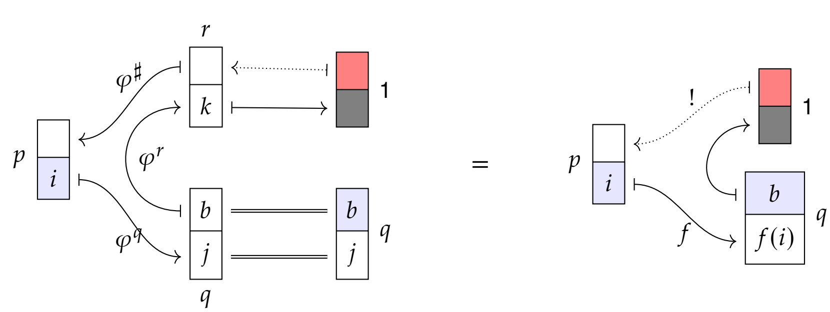
$$\mathbf{Poly}/(q \triangleleft 1)\left(p, q \triangleleft r\right) \cong \mathbf{Poly}\left(p \overset{f}{\frown} q, r\right), \tag{6.74}$$

where

$$p \overset{f}{\frown} q \coloneqq \sum_{i \in p(1)} q[f(i)] y^{p[i]}. \tag{6.75}$$

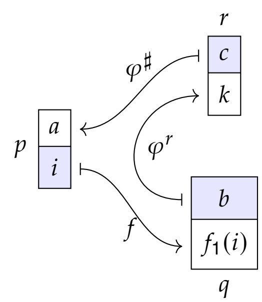
Proof. Again, we present an argument using polyboxes; we leave it to the reader to write this proof in more standard mathematical notation in Exercise 6.77.

Note that to consider $q \triangleleft r$ as an object in $\mathbf{Poly}/(q \triangleleft 1)$, we are implicitly using the lens $q \triangleleft !\colon q \triangleleft r \to q \triangleleft 1$. By definition, morphisms from f to $q \triangleleft !$ in $\mathbf{Poly}/(q \triangleleft 1)$ are lenses $\varphi\colon p \to q \triangleleft r$ for which $\varphi \mathbin{⨾} (q \triangleleft !) = f$. We can write this equation using polyboxes:

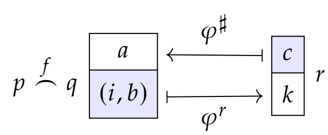


We can read off the picture that a lens $\varphi\colon p \to q \triangleleft r$ is a morphism from f to $q \triangleleft !$ in

Poly$/q \triangleleft 1$ if and only if $\varphi^q = f_1$. So morphisms from f to $q \triangleleft !$ are equivalent to gadgets



with f_1 fixed. But this, in turn, is equivalent to the following gadget (to visualize this equivalence, imagine leaving the polyboxes for r, the arrow $\varphi^\sharp$, and the directions box for p untouched, while dragging the polyboxes for q leftward to the positions box for p, merging the data from q and the predetermined arrow f into a single positions box and adapting the arrow φ^r into an on-positions arrow):



Here the user can provide both the p-position i and the $q[f(i)]$-direction b right from the start, as they know what to expect from f ahead of time. Then the on-positions function encodes the behavior of φ^r. So the polyboxes on the left represent a polynomial whose positions are pairs (i, b) with $i \in p(1)$ and $b \in q[f(i)]$, and whose directions at (i, b) are the directions of the original polynomial p at i. This is precisely the polynomial we defined in (6.75), so the isomorphism holds. □

Remark 6.76. As a lens $p \to q \triangleleft 1$ can be identified with its on-positions function $p(1) \to q(1)$, we'll use the notation $p \overset{f}{\frown} q$ interchangeably for lenses $f \colon p \to q \triangleleft 1$ and functions $f \colon p(1) \to q(1)$.

Exercise 6.77 (Solution here).

1. Translate the polyboxes proof of Proposition 6.73 into standard mathematical notation.
2. Prove that the following natural isomorphism holds:

$$\mathbf{Poly}(p, q \triangleleft r) \cong \sum_{f \colon p(1) \to q(1)} \mathbf{Poly}\left(p \overset{f}{\frown} q, r\right). \tag{6.78}$$

   Thus the functor $(q \triangleleft -) \colon \mathbf{Poly} \to \mathbf{Poly}$ is said to have a *left multiadjoint*.

◊

Exercise 6.79 (Solution here). In stating Proposition 6.73, we implicitly assumed that $p \overset{f}{\frown} q \in \mathbf{Poly}$ is functorial in each variable: covariantly on the left and contravariantly on the right. Here we show that this is indeed the case.

1. Given lenses $f\colon p \to q \triangleleft 1$ and $g\colon p' \to p$, to what lens $p' \overset{g \mathbin{⨟} f}{\frown} q \to p \overset{f}{\frown} q$ should the covariant functor $- \overset{f}{\frown} q$ send g? Prove that your construction is functorial.
2. Given lenses $f\colon p \to q \triangleleft 1$ and $h\colon q \to q'$, to what lens $p \overset{f \mathbin{⨟} (h \triangleleft 1)}{\frown} q' \to p \overset{f}{\frown} q$ should the contravariant functor $p \overset{f}{\frown} -$ send h? Prove that your construction is functorial. ◊

Theorem 6.80 (Preservation of connected limits)**.** The operation $\triangleleft$ preserves connected limits on both sides. That is, if $\mathcal{J}$ is a connected category, $p\colon \mathcal{J} \to \mathbf{Poly}$ is a functor, and $q \in \mathbf{Poly}$ is a polynomial, then there are natural isomorphisms

$$\left(\lim_{j\in\mathcal{J}} p_j\right) \triangleleft q \cong \lim_{j\in\mathcal{J}}(p_j \triangleleft q) \qquad \text{and} \qquad q \triangleleft \left(\lim_{j\in\mathcal{J}} p_j\right) \cong \lim_{j\in\mathcal{J}}(q \triangleleft p_j)$$

Proof. The claim for the left side is just a special case of Proposition 6.68; it remains to prove the claim on the right.

By Theorem 5.33, **Poly** is complete, so by [nLa19, Theorem 4.3], it suffices to show that the functor $(q \triangleleft -)\colon \mathbf{Poly} \to \mathbf{Poly}$ preserves wide pullbacks on the right. By Proposition 6.73, the functor $(q \triangleleft -)\colon \mathbf{Poly} \to \mathbf{Poly}/q \triangleleft 1$ is a right adjoint, so it preserves limits, including wide pullbacks. Thus $(q \triangleleft -)$ sends a wide pullback over $r \in \mathbf{Poly}$ to a wide pullback over the canonical lens $q \triangleleft r \to q \triangleleft 1$ in $\mathbf{Poly}/q \triangleleft 1$, corresponding to a limit in **Poly** of a diagram consisting of arrows to $q \triangleleft r$ and arrows to $q \triangleleft 1$ factoring through $q \triangleleft r$. So up to isomorphism, this limit is just a wide pullback in **Poly** over $q \triangleleft r$, namely $(q \triangleleft -)\colon \mathbf{Poly} \to \mathbf{Poly}$ applied to the original wide pullback. So $\triangleleft$ preserves wide pullbacks on the right. □

Exercise 6.81 (Solution here). Use Theorem 6.80 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:
$$p \triangleleft (qr) \cong (p \triangleleft q) \times_{(p \triangleleft 1)} (p \triangleleft r). \tag{6.82}$$
3. Take the polynomials p, q, r from the counterexample you found in Exercise 6.56 #1 and check that (6.82) holds. ◊

While we're here, it will be helpful to record the following.

Proposition 6.83. For any polynomial $q \in \mathbf{Poly}$, tensoring with q (on either side) preserves connected limits. That is, if $\mathcal{J}$ is connected and $p \colon \mathcal{J} \to \mathbf{Poly}$ is a functor, then there is a natural isomorphism:

$$\left(\lim_{j \in \mathcal{J}} p_j\right) \otimes q \cong \lim_{j \in \mathcal{J}} (p_j \otimes q).$$

6.3.4 Interaction with parallel products

Before we get into how $\otimes$ interacts with $\triangleleft$, here is a warm-up exercise.

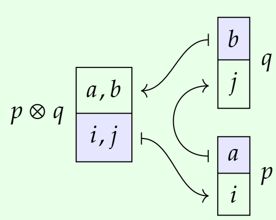
Exercise 6.84 (Solution here). Let A and B be arbitrary sets, and let p be an arbitrary polynomial. Which of the following isomorphisms always hold?

If the isomorphism does not always hold, is there still a canonical lens in one direction or the other?

1. $(Ay) \otimes (By) \cong^? (Ay) \triangleleft (By)$.
2. $y^A \otimes y^B \cong^? y^A \triangleleft y^B$.
3. $A \otimes B \cong^? A \triangleleft B$.
4. $By \otimes p \cong^? By \triangleleft p$.
5. $y^A \otimes p \cong^? y^A \triangleleft p$.
6. $p \otimes By \cong^? p \triangleleft By$.
7. $p \otimes y^A \cong^? p \triangleleft y^A$. ◊

What do all of the lenses you found in this exercise have in common (whether or not they were isomorphisms)?

Example 6.85 (Lenses from $\otimes$ to $\triangleleft$). For any p and q, there is an interesting cartesian lens $o_{p,q} \colon p \otimes q \to p \triangleleft q$ that, stated informally, "orders" the operation, taking the symmetric monoidal product $\otimes$ and reinterprets it as a special case of the asymmetric monoidal product $\triangleleft$. Defining this lens in the usual way is rather tedious and unilluminating, but written in polyboxes, the lens looks like this (recall that positions of $p \otimes q$ are just pairs of positions of p and q, while directions at each such pair are pairs of directions of p and q, one at each position in the pair; we drop the parentheses around the ordered pair for readability):



Usually, the positions box of q is allowed to depend on the directions box of p in the polyboxes for $p \triangleleft q$ on its own. But in the polyboxes above, j is not allowed to depend on a in $p \otimes q$ on the left, so the arrow from the positions box of q to the directions box of p on the right doesn't actually take a into account at all. So the lens $o_{p,q}$ is in some sense the inclusion of the order-independent positions of $p \triangleleft q$; when drawn as trees, the positions in its image are the ones whose upper-level corollas are all the same. And of course we can flip the order using the symmetry $q \otimes p \cong p \otimes q$. This is, we just as well have a lens $p \otimes q \to q \triangleleft p$.

Both $\otimes$ and $\triangleleft$ have the same monoidal unit, the identity functor y, whose identity is the unique lens $y \to y$. In fact the lenses $o_{p,q}$ constitute a lax monoidal functor $(\mathbf{Poly}, y, \otimes) \to (\mathbf{Poly}, y, \triangleleft)$. In particular, $o_{p,q}$ commutes with associators and unitors.

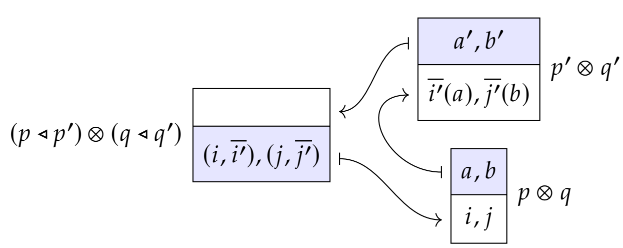
This can be used in the following way. Lenses $p \to q \triangleleft r$ into composites are fairly easy to understand (through polyboxes, for example), whereas lenses $q \triangleleft r \to p$ are not so easy to think about. However, given such a lens, one may always compose it with $o_{q,r}$ to obtain a lens $q \otimes r \to p$. This is quite a bit simpler to think about: they are our familiar interaction patterns from Section 4.4.

It turns out that the monoidal structures $\otimes$ and $\triangleleft$ together satisfy an interesting property known as *duoidality*. We won't give the entire definition of what it means for two monoidal structures to be duoidal here—there are a few commutative diagrams to verify for technical reasons—but the key condition is that there is a natural lens

$$(p \triangleleft p') \otimes (q \triangleleft q') \to (p \otimes q) \triangleleft (p' \otimes q'). \tag{6.86}$$

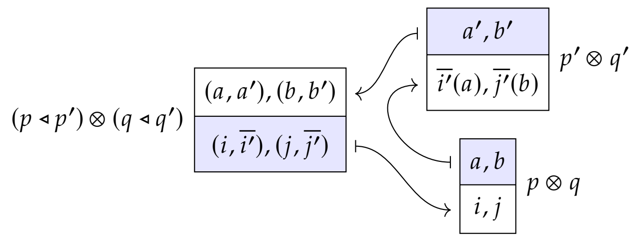
Proposition 6.87. The monoidal structures $(y, \otimes)$ and $(y, \triangleleft)$ together comprise a duoidal structure on **Poly**.

Idea of proof. The key is to give the natural lens from (6.86), as follows. A position of $p \triangleleft p'$ is a pair $(i, \overline{i'})$ with $i \in p(1)$ and $\overline{i'}\colon p[i] \to p'(1)$; similarly, a position $q \triangleleft q'$ is a pair $(j, \overline{j'})$ with $j \in q(1)$ and $\overline{j'}\colon q[j] \to q'(1)$. So we can define the first two parts of the lens using polyboxes, like so (again we drop parentheses around some ordered pairs for readability):



Here (a, b) is a direction of $p \otimes q$ at (i, j), with $a \in p[i]$ and $b \in q[j]$.

Then to fill in the remaining empty box, we need a direction of $p \triangleleft p'$ at $(i, \overline{i'})$, which can be given by the $p[i]$-direction a followed by a $p'[\overline{i'}(a)]$-direction, namely a'. We also need a direction of $q \triangleleft q'$, which can be obtained analogously:



□

6.3.5 Interaction with vertical and cartesian lenses

We conclude this section by examining how $\triangleleft$ interacts with vertical and cartesian lenses, as defined in Definition 5.51.

Proposition 6.88 (Preservation of cartesian lenses)**.** If $\varphi\colon p \to p'$ and $\psi\colon q \to q'$ are cartesian lenses, then so is $\varphi \triangleleft \psi\colon p \triangleleft q \to p' \triangleleft q'$.

Proof. We use the third characterization of cartesian lenses given in Proposition 5.59, as lenses whose naturality squares are pullbacks. For any sets A, B and function $h\colon A \to B$, consider the diagram

$$\begin{array}{ccccc} p \triangleleft q \triangleleft A & \longrightarrow & p' \triangleleft q \triangleleft A & \longrightarrow & p' \triangleleft q' \triangleleft A \\ \downarrow & & \downarrow & & \downarrow \\ p \triangleleft q \triangleleft B & \longrightarrow & p' \triangleleft q \triangleleft B & \longrightarrow & p' \triangleleft q' \triangleleft B. \end{array}$$

The square on the left is a pullback because $\varphi\colon p \to p'$ is cartesian. Meanwhile, the square on the right is a pullback because $\psi\colon q \to q'$ is cartesian and $\triangleleft$ preserves pullbacks by Theorem 6.80. Hence the outer rectangle is a pullback as well, implying that $\varphi \triangleleft \psi\colon p \triangleleft q \to p' \triangleleft q'$ is cartesian. □

Exercise 6.89 (Solution here). Let $\varphi\colon p \to p'$ and $\psi\colon q \to q'$ be lenses.

1. Show that if φ is an isomorphism and ψ is vertical, then $\varphi \triangleleft \psi$ is vertical.
2. Find a vertical lens φ and a polynomial q for which $\varphi \triangleleft q\colon p \triangleleft q \to p' \triangleleft q$ is not vertical. ◊

6.4 Summary and further reading

In this chapter we introduced the composition (sometimes called "substitution") product $\triangleleft$. Given polynomials p, q thought of as functors, their composite is again polynomial and is given by $p \triangleleft q$. We explained how it looks in terms of algebra, e.g. how to compute $y^2 \triangleleft (y + 1)$; in terms of sets, e.g. as a sum-product-sum-product $\sum \prod \sum \prod$, which can be reduced to a single $\sum\prod$; in terms of trees, by stacking corollas on top of corollas; and in terms of polyboxes. We paid particular attention to lenses into a composite $p \to q \triangleleft r$.

This allowed us to explain how to think of dynamical systems $Sy^S \to p \triangleleft q$ with composite interfaces as multi-step machines: each state produces a p-position i, and then for every $p[i]$-direction produces a q-position j, and finally for every $q[j]$-direction returns an updated state in S.

Finally, we discussed some facts of the composition product. For example, we showed that $- \triangleleft q$ has a left adjoint $\left[\frac{q}{-}\right]$ and that $q \triangleleft -$ has a left multi-adjoint $- \frown q$. We also explained that $p \triangleleft q$ preserves all with limits in the variable p and all connected limits in the variable q. We also explained the duoidal interaction between $\otimes$ and $\triangleleft$, i.e. the natural lens $\triangleleft \otimes \triangleleft \to \otimes \triangleleft \otimes$, and how $\triangleleft$ interacts with cartesian lenses.

Polynomial substitution is one of the best known aspects of polynomial functors. Again, see [GK12] for more on this. We learned of the left coclosure (see Proposition 6.57) from Josh Meyers, though it may have already been known in the containers community.

6.5 Exercise solutions

Solution to Exercise 6.8.

We are given $p := y^2 + y^1$ and $q := y^3 + 1$.

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 $\overline{j_1}\colon p[1] \to q(1)$ is one of the two possible bijections, we have that
$$\sum_{a \in p[1]} q[\overline{j_1}(a)] \cong q[1] + q[2] \cong 3 + 0 \cong 3.$$
When $\overline{j_1}\colon p[1] \to q(1)$ sends everything to $1 \in q(1)$, we have that
$$\sum_{a \in p[1]} q[\overline{j_1}(a)] \cong q[1] + q[1] \cong 3 + 3 \cong 6.$$
Finally, when $\overline{j_1}\colon p[1] \to q(1)$ sends everything to $2 \in q(1)$, we have that
$$\sum_{a \in p[1]} q[\overline{j_1}(a)] \cong q[2] + q[2] \cong 0 + 0 \cong 0.$$
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 $\sum_{a \in p[2]} q[\overline{j_2}(a)] \cong q[1] \cong 3$, and when $\overline{j_2}\colon p[2] \to q(1)$ maps to $2 \in q(1)$, we have that $\sum_{a \in p[2]} q[\overline{j_2}(a)] \cong q[2] \cong 0$.
8. From the previous parts, we have that
$$\sum_{i \in p(1)} \sum_{\overline{j}\colon p[i] \to q(1)} y^{\sum_{a \in p[i]} q[\overline{j_i}(a)]} \cong (2y^3 + y^6 + y^0) + (y^3 + y^0) \cong y^6 + 3y^3 + 2,$$
which agrees with what $p \triangleleft q$ should be.

Solution to Exercise 6.9.

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 6.28).

Solution to Exercise 6.10.

Given $A \in \mathbf{Set}$ and $q \in \mathbf{Poly}$, we have

$$\begin{aligned} y^A \triangleleft q &\cong \sum_{\bar{j}\colon A \to q(1)} y^{\sum_{a \in A} q[\bar{j}(a)]} && (6.7) \\ &\cong \sum_{\varphi\colon Ay \to q} y^{\sum_{a \in A} q[\varphi_1(a)]} \\ &\cong [Ay, q], && (4.79) \end{aligned}$$

for a lens $\varphi\colon Ay \to q$ has an on-positions function $A \to q(1)$ and uniquely determined on-directions functions.

Solution to Exercise 6.13.

We wish to show that (6.14) could replace (6.12) in Definition 6.11. We claim that (6.12) and (6.14) 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 6.18.

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 a:
      2.1. choose a p-position i';
      2.2. for each $p[i']$-direction a':
         2.2.1. choose a p-position i'';
         2.2.2. for each $p[i'']$-direction a'':
            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 a:
      2.1. choose a p-position i';
      2.2. for each $p[i']$-direction a':
         2.2.1. done.

Solution to Exercise 6.19.

We have lenses $f\colon p \to p'$ and $g\colon q \to q'$.

1. By Proposition 3.44, 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 $(f_1(i), f_i^\sharp \mathbin{⨟} h)$. 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 $a \in p[i]$, a function $h_a\colon q[\bar{j}_i(a)] \to X$. So $f_{q(X)}\colon (p \triangleleft q)(X) \to (p' \triangleleft q)(X)$ sends

$$(i, \bar{j}_i, (h_a)_{a\in p[i]}) \mapsto \left(f_1(i), f_i^\sharp \mathbin{⨾} \bar{j}_i, \left(h_{f_i^\sharp(a')}\right)_{a'\in p'[f_1(i)]}\right).$$

2. By Proposition 3.44, 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 $(g_1(j), g_j^\sharp \mathbin{⨾} k)$ in $q'(X)$. Then by Proposition 2.10, applying p' to this X-component yields a function $p'(q(X)) \to p'(q'(X))$ that sends every $(i', \bar{j'}_{i'}, (h'_{a'})_{a'\in p'[i']})$ with $i' \in p'(1)$ as well as $\bar{j'}_{i'}\colon p'[i'] \to q(1)$ and $h'_{a'}\colon q[\bar{j'}_{i'}(a')] \to X$ to

$$\left(i', \bar{j'}_{i'} \mathbin{⨾} g_1, \left(g^\sharp_{\bar{j'}_{i'}(a')} \mathbin{⨾} h'_{a'}\right)_{a'\in p'[i']}\right).$$

3. By Definition 6.11, the horizontal composite of f and g is the natural transformation $f \triangleleft g\colon p \triangleleft p' \to q \triangleleft q'$ whose X-component is the composite of the answers to #1 and #2, sending

$$\begin{aligned}(i, \bar{j}_i, (h_a)_{a\in p[i]}) &\mapsto \left(f_1(i), f_i^\sharp \mathbin{⨾} \bar{j}_i, \left(h_{f_i^\sharp(a')}\right)_{a'\in p'[f_1(i)]}\right)\\ &\mapsto \left(f_1(i), f_i^\sharp \mathbin{⨾} \bar{j}_i \mathbin{⨾} g_1, \left(g^\sharp_{\bar{j}_i(f_i^\sharp(a'))} \mathbin{⨾} h_{f_i^\sharp(a')}\right)_{a'\in p'[f_1(i)]}\right).\end{aligned}$$

4. We use Corollary 3.47 to translate the answer to #3 into a lens $f \triangleleft g\colon p \triangleleft q \to p' \triangleleft q'$, as follows. Its on-positions function is the 1-component $(f \triangleleft g)_1$, which sends every $(i, \bar{j}_i)$ with $i \in p(1)$ and $\bar{j}_i\colon p[i] \to q(1)$ to

$$(f_1(i), f_i^\sharp \mathbin{⨾} \bar{j}_i \mathbin{⨾} g_1).$$

Then for each such $(i, \bar{j}_i)$, if we apply the $(p \triangleleft q)[(i, \bar{j}_i)]$-component of $f \triangleleft g$ to the element $(i, \bar{j}_i, (\iota_d)_{a\in p[i]})$, where $\iota_d\colon q[\bar{j}_i(a)] \to (p \triangleleft q)[(i, \bar{j}_i)] \cong \sum_{a\in p[i]} q[\bar{j}_i(a)]$ is the canonical inclusion, then take the last coordinate of the result, we obtain for each $a' \in p'[f_1(i)]$ the function

$$q'[g_1(\bar{j}_i(f_i^\sharp(a')))] \xrightarrow{g^\sharp_{\bar{j}_i(f_i^\sharp(a'))}} q[\bar{j}_i(f_i^\sharp(a'))] \xrightarrow{\iota_{f_i^\sharp(a')}} \sum_{a\in p[i]} q[\bar{j}_i(a)] \cong (p \triangleleft q)[(i, \bar{j}_i)].$$

These can equivalently be thought of as a single function from

$$\sum_{a'\in p'[f_1(i)]} q'[g_1(\bar{j}_i(f_i^\sharp(a')))] \cong (p' \triangleleft q')[(f \triangleleft g)_1(i, \bar{j}_i)]$$

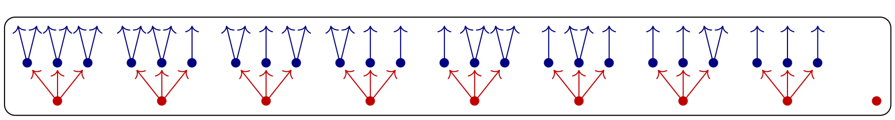
which Corollary 3.47 tells us is the on-directions function of $f \triangleleft g$ at $(i, \bar{j}_i)$, that sends every (a', b') with $a' \in p'[f_1(i)]$ and $b' \in q'[g_1(\bar{j}_i(f_i^\sharp(a')))]$ to

$$\left(f_i^\sharp(a'), g^\sharp_{\bar{j}_i(f_i^\sharp(a'))}(b')\right).$$

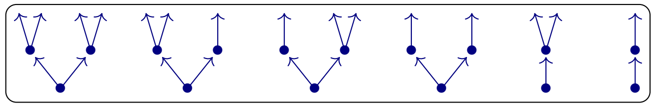
Solution to Exercise 6.26.

We have $p := y^2 + y$ and $q := y^3 + 1$ as in (6.22).

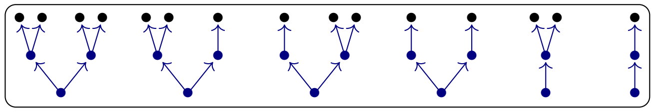
1. Here is a picture of $q \triangleleft p$, where each tree is obtained by taking a q-corolla and grafting p-corollas to every leaf:



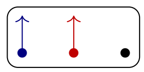
2. Here is a picture of $p \triangleleft p$:



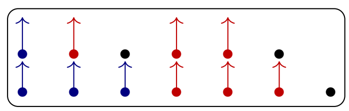
3. To obtain a picture of $p \triangleleft p \triangleleft 1$, we take our picture of $p \triangleleft p$ and graft the single, leafless 1-root onto every (height-2) leaf:



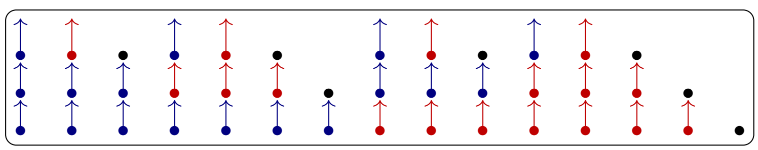
Now $r := 2y + 1$. Before we draw the composites, here's a picture of r itself, with different colors to distinguish the different positions:



4. Here is a picture of $r \triangleleft r$:

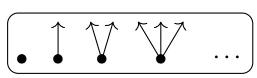


5. Here is a picture of $r \triangleleft r \triangleleft r$:

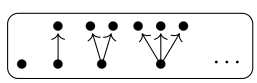


Solution to Exercise 6.28.

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 (6.6):
$$X \triangleleft p \cong \sum_{i \in X} \prod_{a \in \varnothing} \sum_{j \in p(1)} \prod_{b \in p[j]} y \cong \sum_{i \in X} 1 \cong X.$$
3. When viewed as functors, it is easy to see that $p \triangleleft X \cong p(X)$. We can also verify this using (6.6):
$$p \triangleleft X \cong \sum_{i \in p(1)} \prod_{a \in p[i]} \sum_{j \in X} \prod_{b \in \varnothing} y \cong \sum_{i \in p(1)} \prod_{a \in p[i]} \sum_{j \in X} 1 \cong \sum_{i \in p(1)} \prod_{a \in p[i]} X \cong \sum_{i \in p(1)} X^{p[i]} \cong p(X).$$

Solution to Exercise 6.29.

Let $\varphi \colon p \to q$ be a lens and X be a set viewed as a constant polynomial. Note that every component of the identity natural transformation on X as a constant functor is just the identity function $\mathrm{id}_X \colon X \to X$

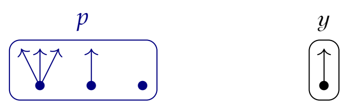
on X as a set. Then by Definition 6.11, any component of the composition product $\varphi \triangleleft X$ viewed as a natural transformation is given by the composite function

$$p(X) \xrightarrow{\varphi_X} q(X) \xrightarrow{q(\mathrm{id}_X)} q(X).$$

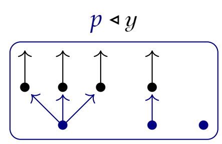
By functoriality, $q(\mathrm{id}_X)$ is itself an identity function, so every component of $\varphi \triangleleft X$ is the X-component of φ. Therefore $\varphi \triangleleft X$ as a function can be identified with the X-component of φ, as desired.

Solution to Exercise 6.30.

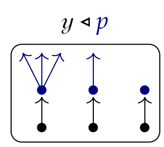
1. The polynomial y is the identity functor on **Set**.
2. Composing any functor with the identity functor yields the original functor, so $p \triangleleft y \cong p \cong y \triangleleft p$.
3. Before we draw $y \triangleleft p$ and $p \triangleleft y$, here are pictures of p and y individually as corolla forests:



Now here is a picture of $p \triangleleft y$, obtained by grafting the one-leaf y-corolla to all the leaves of each p-corolla in turn:

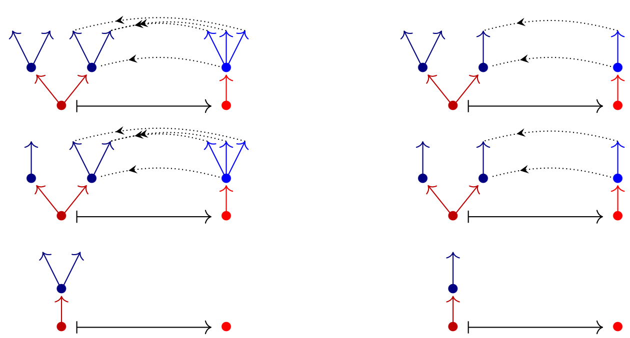


This is just p with every direction extended up one level, so it is still a picture of p.
And here is a picture of $y \triangleleft p$, obtained by grafting each p-corolla to the single leaf of y:



Solution to Exercise 6.32.

Using the definitions, instructions, and style from Example 6.31, we draw $\psi \triangleleft \varphi \colon q \triangleleft p \to q' \triangleleft p'$:



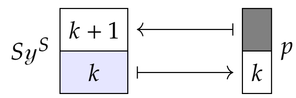
Solution to Exercise 6.33.

Given arbitrary polynomials p, q, r and lenses $\varphi \colon q \to p \triangleleft q$ and $\psi \colon q \to q \triangleleft r$, it is *not* necessarily the case that $\varphi \fatsemi (p \triangleleft \psi) = \psi \fatsemi (\varphi \triangleleft r)$! After all, we can let $p := y$ and $q := 2$ so that φ is a lens $2 \to y \triangleleft 2 \cong 2$ (see Exercise 6.30) and ψ is a lens $2 \to 2 \triangleleft r \cong 2$ (see Exercise 6.28). Then by following the instructions for interpreting a composition product of lenses from either Exercise 6.19 or Example 6.31, we can verify that $p \triangleleft \psi = y \triangleleft \psi$ is a lens $2 \cong y \triangleleft 2 \to y \triangleleft 2 \triangleleft r \cong 2$ equivalent to the lens ψ, while $\varphi \triangleleft r$ is a lens $2 \cong 2 \triangleleft r \to y \triangleleft 2 \triangleleft r \cong 2$ equivalent to the lens φ. If, say, we let $\varphi \colon 2 \to 2$ be the function sending everything to $1 \in 2$ and $\psi \colon 2 \to 2$ be the function sending everything to $2 \in 2$, then in this case $\varphi \fatsemi (p \triangleleft \psi) = \varphi \fatsemi \psi \neq \psi \fatsemi \varphi = \psi \fatsemi (\varphi \triangleleft r)$.

Solution to Exercise 6.45.

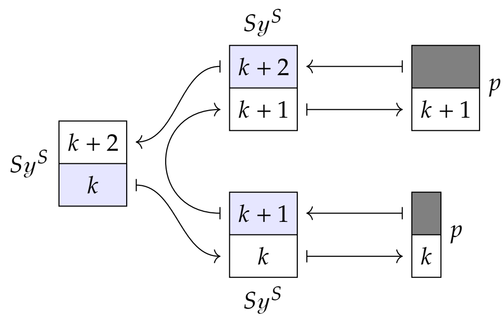
Given $S := \mathbb{N}$ and $p := \mathbb{R}y^1$, we have a dynamical system $\varphi\colon Sy^S \to p$ given by $\varphi_1(k) := k$ and $\varphi^\sharp_k(1) := k+1$.

1. Here is the polybox picture for φ (recall that we shade the upper box of a linear polynomial gray, as there is only one option to place there):



A single run through the system returns the current state $k \in \mathbb{N}$, then increases that state by 1.

2. Here is the polybox picture for $\delta \mathbin{⨟} (\varphi \triangleleft \varphi)\colon Sy^S \to p \triangleleft p$:



The new system has $p \triangleleft p \cong \mathbb{R}y \triangleleft \mathbb{R}y \cong \mathbb{R}^2 y$ as its interface. Indeed, we see that it returns two numbers at once: the current state k (what the first run through φ would return) as well as the increased state $k+1$ (what the second run through φ would return). We update the current state from k to $k+1$ in one run, and from $k+1$ to $k+2$ in the next—thus increasing the current state by 2 overall.

Solution to Exercise 6.46.

We give two reasonable (and of course equivalent) ways to interpret the transition lens $\delta\colon Sy^S \to Sy^S \triangleleft Sy^S$.

One way is to first evaluate its interface as $Sy^S \triangleleft Sy^S \cong S\left(Sy^S\right)^S \cong \left(S \times S^S\right) y^{S\times S}$. Then we see that if the current state is $s_0 \in S$, the system returns a position consisting of that current state s_0 along with a function $S \to S$, namely the identity function $s_1 \mapsto s_1$. The system then takes in a pair $(s_1, s_2) \in S \times S$, discarding s_1 and setting its new state to be s_2.

Alternatively, we can draw from Example 6.41 to interpret δ as a dynamical system that behaves as follows. Each run through the system is a 2-step process: first, the current state $s_0 \in S$ is returned, and a new state $s_1 \in S$ is received. Then this new state s_1 is immediately returned, and an ever newer state $s_2 \in S$ is received. Then the current state is updated to the newer state s_2.

Solution to Exercise 6.52.

To prove Proposition 6.47, it suffices to verify (6.50) and (6.51), as (6.48) and (6.49) follow directly when $A := 2$.

Given polynomials $(q_a)_{a\in A}$, recall that the position-set of the sum $\sum_{a\in A} q_a$ is $\sum_{a\in A} q_a(1)$, while the direction-set at each position (a, j) with $a \in A$ and $j \in q_a(1)$ is $q_a[j]$. So by (6.6), we have that

$$\begin{aligned}\left(\sum_{a\in A} q_a\right) \triangleleft r &\cong \sum_{\substack{a\in A,\\ j\in q_a(1)}} \prod_{b\in q_a[j]} \sum_{k\in r(1)} \prod_{c\in r[k]} y \\ &\cong \sum_{a\in A} \sum_{j\in q_a(1)} \prod_{b\in q_a[j]} \sum_{k\in r(1)} \prod_{c\in r[k]} y\end{aligned}$$

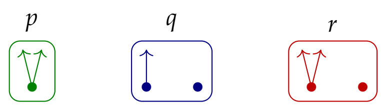
$$\cong \sum_{a \in A} (q_a \triangleleft r).$$

We can also recall that the position-set of the product $\prod_{a \in A} q_a$ is $\prod_{a \in A} q_a(1)$, while the direction-set at each position $\bar{j} : (a \in A) \to q_a(1)$ is $\sum_{a \in A} q_a[\bar{j}(a)]$. So by (6.6), we have that

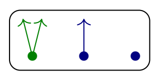
$$\begin{aligned}
\left(\prod_{a \in A} q_a\right) \triangleleft r &\cong \sum_{\bar{j} \in \prod_{a \in A} q_a(1)} \prod_{\substack{a \in A, \\ b \in q_a[\bar{j}(a)]}} \sum_{k \in r(1)} \prod_{c \in r[j]} y \\
&\cong \prod_{a \in A} \sum_{j \in q_a(1)} \prod_{b \in q_a[j]} \sum_{k \in r(1)} \prod_{c \in r[k]} y \qquad (1.32) \\
&\cong \prod_{a \in A} (q_a \triangleleft r).
\end{aligned}$$

Solution to Exercise 6.54.

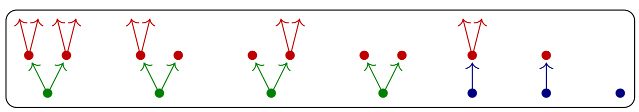
We want an intuitive understanding of the left distributivity of $\triangleleft$ over $+$. Let $p := y^2$, $q := y + 1$, and $r := y^2 + 1$, as shown here:



Then $p + q \cong y^2 + y + 1$ can be drawn as follows, consisting of every p-corolla and every q-corolla:

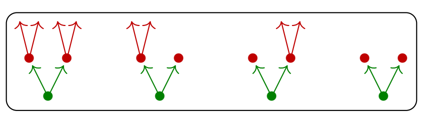
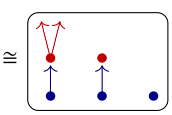
$p + q \cong$ 

We can therefore draw $(p + q) \triangleleft r$ by grafting r-corollas to leaves of $p + q$ in every way, as follows:

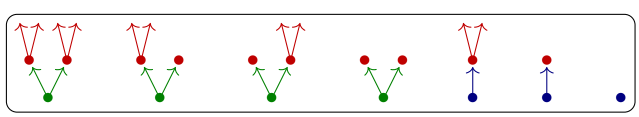
$(p + q) \triangleleft r \cong$ 

So each tree in $(p + q) \triangleleft r$ is obtained by taking either a p-corolla or a q-corolla, then attaching an r-corolla to each leaf.

Alternatively, we can compute $p \triangleleft r$ and $q \triangleleft r$ separately, grafting r-corollas 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 p-corolla with an r-corolla attached to each leaf, or a q-corolla with an r-corolla attached to each leaf.

But it doesn't matter whether we graft r-corollas onto 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 6.55.

Given a set A and polynomials p, q, the left distributivity of $\triangleleft$ over products from (6.51) implies that $(Ap) \triangleleft q \cong (A \triangleleft q)(p \triangleleft q)$, while Exercise 6.28 #3 implies that $A \triangleleft q \cong A$. So $(Ap) \triangleleft q \cong A(p \triangleleft q)$.

Solution to Exercise 6.56.

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 6.60.

We prove Proposition 6.57 by observing that

$$\begin{aligned}
\mathbf{Poly}(p, r \triangleleft q) &\cong \prod_{i \in p(1)} r(q(p[i])) && (3.7)\\
&\cong \mathbf{Poly}\left(\sum_{i \in p(1)} y^{q(p[i])}, r\right) && (3.7)\\
&\cong \mathbf{Poly}\left(\begin{bmatrix} q \\ p \end{bmatrix}, r\right). && (6.59)
\end{aligned}$$

Solution to Exercise 6.62.

1. Given a polynomial q and a lens $\varphi \colon p \to p'$, the functor $\begin{bmatrix} q \\ - \end{bmatrix}$ should send φ to a lens $\begin{bmatrix} q \\ p \end{bmatrix} \to \begin{bmatrix} q \\ p' \end{bmatrix}$; by (6.59), this is a lens
$$\begin{bmatrix} q \\ \varphi \end{bmatrix} \colon \sum_{i \in p(1)} y^{q(p[i])} \to \sum_{i' \in p'(1)} y^{q(p'[i'])}.$$
We give $\begin{bmatrix} q \\ \varphi \end{bmatrix}$ the same on-positions function $p(1) \to p'(1)$ that φ has. Then viewing q as a functor, we define the on-directions function $\begin{bmatrix} q \\ \varphi \end{bmatrix}_i^\sharp \colon q(p'[\varphi_1(i)]) \to q(p[i])$ for each $i \in p(1)$ to be the function obtained by applying q to the corresponding on-directions function $\varphi_i^\sharp \colon p'[\varphi_1(i)] \to p[i]$. Functoriality follows trivially on positions and by the functoriality of q itself on directions.
2. Given a polynomial p and a lens $\psi \colon q' \to q$, the functor $\begin{bmatrix} - \\ p \end{bmatrix}$ should send ψ to a lens $\begin{bmatrix} q \\ p \end{bmatrix} \to \begin{bmatrix} q' \\ p \end{bmatrix}$; by (6.59), this is a lens
$$\begin{bmatrix} \psi \\ p \end{bmatrix} \colon \sum_{i \in p(1)} y^{q(p[i])} \to \sum_{i \in p(1)} y^{q'(p[i])}.$$
We let the on-positions function of $\begin{bmatrix} \psi \\ p \end{bmatrix}$ be the identity on $p(1)$. Then viewing ψ as a natural transformation, we define the on-directions function $\begin{bmatrix} \psi \\ p \end{bmatrix}_i^\sharp \colon q'(p[i]) \to q(p[i])$ for each $i \in p(1)$ to be the $p[i]$-component of ψ. Functoriality follows trivially on positions and because natural transformations compose componentwise on directions.

Solution to Exercise 6.3.2.

In order for $\begin{bmatrix} q \\ p \end{bmatrix}$ to be a left Kan extension of p along q, we need to first provide a natural transformation $p \to \begin{bmatrix} q \\ p \end{bmatrix} \triangleleft q$, and second show that it is universal. But this is exactly the content of Proposition 6.57: the adjunction (6.58) provides a unit $p \to \begin{bmatrix} q \\ p \end{bmatrix} \triangleleft q$ that is universal in the appropriate way.

Solution to Exercise 6.64.

We are given $A, B \in \mathbf{Set}$ and $p, q \in \mathbf{Poly}$

1. By (6.59),
$$\begin{bmatrix} B \\ A \end{bmatrix} = \sum_{i \in A} y^B \cong Ay^B,$$
so by (6.58),
$$\mathbf{Poly}(Ay^B, p) \cong \mathbf{Poly}(A, p \triangleleft B).$$
But A and $p \triangleleft B \cong p(B)$ are both constants, so a lens $A \to p \triangleleft B$ is just a function $A \to p(B)$ on positions. Hence (6.65) follows.

2. We prove (6.66) in two parts: that
$$\mathbf{Poly}\left(Ay \triangleleft p, q\right) \cong \mathbf{Poly}\left(p, y^A \triangleleft q\right) \tag{6.90}$$
and that
$$\mathbf{Poly}\left(p \triangleleft y^B, q\right) \cong \mathbf{Poly}\left(p, q \triangleleft By\right). \tag{6.91}$$
We have that $Ay \triangleleft p \cong Ap$ and $y^A \triangleleft q \cong q^A$, so (6.90) follows from (5.9). Meanwhile, for (6.91), we have by (6.59) that
$$\begin{bmatrix} By \\ p \end{bmatrix} = \sum_{i \in p(1)} y^{Bp[i]} \cong p \triangleleft y^B,$$
so (6.66) follows from (6.58). Then combining (6.90) and (6.91) yields
$$\mathbf{Poly}\left(Ay \triangleleft p \triangleleft y^B, q\right) \cong \mathbf{Poly}\left(p \triangleleft y^B, y^A \triangleleft q\right) \cong \mathbf{Poly}\left(p, y^A \triangleleft q \triangleleft By\right).$$

Solution to Exercise 6.70.

1. We wish to solve Exercise 6.28 #3 using (6.69) and (6.50). If we set $\mathcal{J}$ in (6.69) 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 (6.50),
$$X \triangleleft p \cong \left(\sum_{x \in X} 1\right) \triangleleft p \cong \sum_{x \in X} (1 \triangleleft p) \cong \sum_{x \in X} 1 \cong X.$$

2. If we set $\mathcal{J}$ in (6.69) 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 (6.51) 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 6.77.

1. Note that $\mathbf{Poly}(p, q \triangleleft 1) \cong \mathbf{Set}(p(1), q(1))$. A lens $p \to q \triangleleft r$ is an element of
$$\prod_{i \in p(1)} \sum_{j \in q(1)} \prod_{b \in q[j]} \sum_{k \in r(1)} \prod_{c \in r[k]} p[i] \cong \sum_{f\colon p(1) \to q(1)} \prod i \in p(1) \prod_{b \in q[j]} \sum_{k \in r(1)} \prod_{c \in r[k]} p[i]$$
Fixing the function $f\colon p(1) \to q(1)$ as implicit in p, we get
$$\begin{aligned} \mathbf{Poly}/(q \triangleleft 1)\left(p, q \triangleleft r\right) &\cong \prod i \in p(1) \prod_{b \in q[j]} \sum_{k \in r(1)} \prod_{c \in r[k]} p[i] \\ &\cong \mathbf{Poly}\left(p \overset{f}{\frown} q, r\right), \end{aligned}$$

2. This follows from the first isomorphism above.

Solution to Exercise 6.79.

1. Given lenses $f\colon p \to q \triangleleft 1$ and $g\colon p' \to p$, the functor $- \overset{f}{\frown} q$ should send g to a lens $p' \overset{g \mathbin{⨾} f}{\frown} q \to p \overset{f}{\frown} q$; by (6.75), this is a lens
$$g \overset{f}{\frown} q\colon \sum_{i' \in p'(1)} q[f_1(g_1(i'))] y^{p'[i']} \to \sum_{i \in p(1)} q[f_1(i)] y^{p[i]}.$$

We give $g \overset{f}{\frown} q$ an on-positions function that is the on-positions function $p'(1) \to p(1)$ of g on the first coordinate $i' \in p'(1)$ and the identity on $q[f_1(g_1(i'))]$ on the second. Then we let the on-directions function at every position with first coordinate $i' \in p'(1)$ be the on-directions function $g^\sharp_{i'}\colon p[g_1(i')] \to p'[i']$. Functoriality follows trivially on both positions and directions.

2. Given a polynomial p and lenses $f\colon p \to q \triangleleft 1$ and $h\colon q \to q'$, the functor $p \overset{f}{\frown} -$ should send h to a lens $p \overset{f \mathbin{⨾} (h \triangleleft 1)}{\frown} q' \to p \overset{f}{\frown} q$; by (6.75), this is a lens

$$p \overset{f}{\frown} h\colon \sum_{i \in p(1)} q'[h_1(f_1(i))]y^{p[i]} \to \sum_{i \in p(1)} q[f_1(i)]y^{p[i]}.$$

We let the on-positions function of $p \overset{f}{\frown} h$ be the identity on the first coordinate $i \in p(1)$ and the on-directions function $h^\sharp_{f_1(i)}\colon q'[h_1(f_1(i))] \to q[f_1(i)]$ on the second. Then we let the on-directions function at every position with first coordinate $i \in p(1)$ be the identity on $p[i]$. Functoriality follows trivially on both positions and directions.

Solution to Exercise 6.81.

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 $\mathcal{J}$ and a functor $X\colon \mathcal{J} \to \mathbf{Set}$, we have

$$p\left(\lim_{j \in \mathcal{J}} X_j\right) \cong \lim_{j \in \mathcal{J}} p(X_j).$$

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 6.28 #3, the left hand side of the isomorphism we seek is isomorphic to $p \triangleleft \left(\lim_{j \in \mathcal{J}} X_j\right)$, while the right hand side is isomorphic to $\lim_{j \in \mathcal{J}} \left(p \triangleleft X_j\right)$. These are isomorphic by Theorem 6.80.

2. Given $p, q, r \in \mathbf{Poly}$, we wish to show that (6.82) holds. 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 6.80, they are preserved by precomposition with p. Hence the desired isomorphism follows.

3. We'll show that (6.82) holds for $p \coloneqq y + 1$, $q \coloneqq 1$, and $r \coloneqq 0$. We have $p \triangleleft q = p \triangleleft 1 \cong (y+1) \triangleleft 1 \cong 2$ and $p \triangleleft r \cong (y+1) \triangleleft 0 \cong 1$, so $(p \triangleleft q) \times_{(p \triangleleft 1)} (p \triangleleft r) \cong 2 \times_2 1$. We saw in Example 5.38 that the position-set of a pullback in **Poly** is just the pullback of the position-sets in **Set**, while the direction-sets are given by a pushout of direction-sets in **Set**. As our polynomials have empty direction-sets, their pullback must have an empty direction-set as well, so this pullback is just a pullback of sets: $(p \triangleleft q) \times_{(p \triangleleft 1)} (p \triangleleft r) \cong 2 \times_2 1 \cong 1$. And indeed we have $p \triangleleft (qr) \cong (y+1) \triangleleft 0 \cong 1$ as well.

Solution to Exercise 6.84.

Here A and B are sets and p is a polynomial.

1. The isomorphism always holds: we have that $(Ay) \otimes (By) \cong ABy \cong (Ay) \triangleleft (By)$.
2. The isomorphism always holds: we have that $y^A \otimes y^B \cong y^{AB} \cong y^A \triangleleft y^B$.
3. The isomorphism does not always hold: while $A \otimes B \cong AB$, we have that $A \triangleleft B \cong A$. There is, however, always a canonical projection $AB \to B$; but there is not always a canonical lens $B \to AB$ (for example, take $A = 0 \neq B$).
4. The isomorphism always holds: we have that $By \otimes p \cong \sum_{i \in p(1)} By \otimes y^{p[i]} \cong \sum_{i \in p(1)} By^{p[i]} \cong Bp \cong By \triangleleft p$.
5. The isomorphism does not always hold: if, say, $p = B$, then $y^A \otimes B \cong B$, while $y^A \triangleleft B \cong B^A$. If $A = B = 0$, then $B^A \cong 1$, so there is not always a canonical lens from right to left, either. There is, however, always a canonical lens from left to right: $y^A \otimes p \cong \sum_{i \in p(1)} y^{Ap[i]}$ while $y^A \triangleleft p \cong \sum_{\bar{i}\colon A \to p(1)} y^{\sum_{a \in A} p[\bar{i}(a)]}$. So there is a lens from left to right whose on-positions function sends $i \in p(1)$ to the constant function $A \to p(1)$ that always evaluates to i; and whose on-directions function at i is the identity on $Ap[i]$.
6. The isomorphism does not always hold: if, say, $p = y^A$, then $y^A \otimes By \cong By^A$, while $y^A \otimes By \cong (By)^A \cong B^A y^A$. If $A = B = 0$, then $By^A \cong 0$ while $B^A y^A \cong 0^0 y^0 \cong 1$, so there is not always a

canonical lens from right to left, either. There is, however, always a canonical lens from left to right: $p \otimes By \cong Bp$ while $p \triangleleft By \cong \sum_{i \in p(1)} \sum_{\overline{b} \colon p[i] \to B} y^{\sum_{a \in p[i]} 1} \cong \sum_{i \in p(1)} B^{p[i]} y^{p[i]}$. So there is a lens from left to right whose on-positions function sends $(b, i) \in Bp(1)$ to $(i, c_b) \in \sum_{i \in p(1)} B^{p[i]}$, where $c_b \colon p[i] \to B$ is the constant function that always evaluates to b; and whose on-directions function at (b, i) is the identity on $p[i]$.

7. The isomorphism always holds: we have that $p \otimes y^A \cong \sum_{i \in p(1)} y^{Ap[i]} \cong p \triangleleft y^A$.

Every on-directions function of every lens we found in this exercise are isomorphisms, so every lens we found in this exercise is cartesian.

Solution to Exercise 6.89.

Here $\varphi \colon p \to p'$ and $\psi \colon q \to q'$ are lenses.

1. If φ is an isomorphism and ψ is vertical, then $\psi \triangleleft 1 \colon q \triangleleft 1 \to q' \triangleleft 1$ is an isomorphism, so $\varphi \triangleleft \psi \triangleleft 1 \colon p \triangleleft q \triangleleft 1 \to p' \triangleleft q' \triangleleft 1$ is an isomorphism as well. Thus $\varphi \triangleleft \psi$ is vertical.
2. If φ is the unique lens $y \to 1$ and $q = 0$, then φ is vertical, but since $y \triangleleft 0 \cong 0$ and $1 \triangleleft 0 \cong 1$, the lens $\varphi \triangleleft 0 \colon 0 \to 1$ is not.

Chapter 7

Polynomial comonoids and retrofunctors

Imagine a 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 the original position 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 presented a characterization of comonoids in $(\mathbf{Poly}, y, \triangleleft)$ as a surprisingly familiar construct. For us, this story will emerge naturally as we continue to expand our understanding of the humble state system of a dependent dynamical system.

7.1 State systems, categorically

Since defining dependent dynamical systems in Definition 4.18, we have evolved our understanding of their state systems over the course of the last few chapters. Let's take this moment to review what we know about these state systems so far.

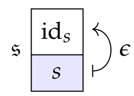
Our original definition of a state system was as a monomial Sy^S for some set S. But in Example 4.43, we noted that this formulation requires us to discuss the positions and directions of a state system at the level of sets rather than in the language of **Poly**. Instead, let's take an arbitrary polynomial $\mathfrak{s} \in \mathbf{Poly}$ and attempt to characterize what it means for $\mathfrak{s}$ to be a state system using only the categorical machinery of **Poly**. We will continue to refer to the positions of $\mathfrak{s}$ as *states*, but we will shift from thinking of the directions of $\mathfrak{s}$ as states to thinking of them as transitions from one state to another.

7.1.1 The do-nothing section

In Example 4.43, we saw that every state system $\mathfrak{s}$ is equipped with a *do-nothing section*: a lens $\epsilon\colon \mathfrak{s} \to y$ that picks out a direction at each state that we would like to interpret as "doing nothing" and remaining at that state.

We drew ϵ in polyboxes in Example 4.45, but that was when we let ourselves assume that the position-set of a state system was equal to each of its direction-sets. Now all we know is that for each state $s \in \mathfrak{s}(1)$, the do-nothing section chooses an $\mathfrak{s}[s]$-direction to signify staying at the same state; it doesn't make sense to say that this direction is literally equal to s. So we need a different name for the $\mathfrak{s}[s]$-direction that ϵ identifies: call it id_s, because it behaves like a sort of identity operation on the state s.

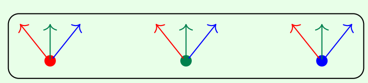
So the revised polyboxes for the do-nothing section $\epsilon\colon \mathfrak{s} \to y$ are as follows:



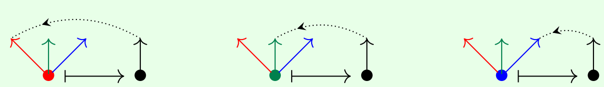
(7.1)

Exercise 7.2 (Solution here). Say I have a polynomial $\mathfrak{s} \in \mathbf{Poly}$, and I tell you that there is a lens $\epsilon\colon \mathfrak{s} \to y$. What can you say about the polynomial $\mathfrak{s}$? $\Diamond$

Example 7.3 (The do-nothing section in tree pictures). We have seen the do-nothing section drawn in polyboxes, but let's see what it looks like in our tree pictures. We'll take $\mathfrak{s} := \{{\color{red}\bullet}, {\color{green}\bullet}, {\color{blue}\bullet}\}y^{\{{\color{red}\bullet},{\color{green}\bullet},{\color{blue}\bullet}\}}$, drawn as follows:

$\mathfrak{s} :=$ 

Then the do-nothing section $\epsilon\colon \mathfrak{s} \to y$ can be drawn like any one of the following three possibilities:

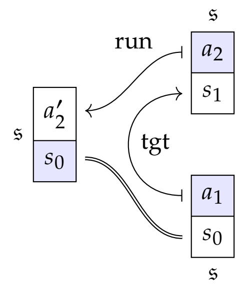


It picks out one direction at each position, namely the one of the same color.

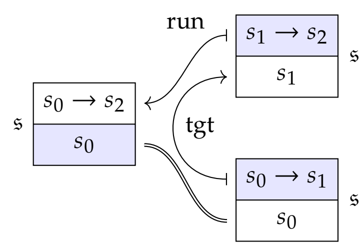
There is not much else we can say about the do-nothing section on its own, so let us revisit the other lens that every state system is equipped with before considering the relationship between the two.

7.1.2 The transition lens

We saw in Example 6.44 that $\mathfrak{s}$ also comes equipped with a *transition lens*: a lens $\delta\colon \mathfrak{s} \to \mathfrak{s} \triangleleft \mathfrak{s}$, which we can draw as



The arrow labeled tgt is the *target function*: given a state s_0 and a direction a_1 at that state, $\mathrm{tgt}(s_0, a_1)$ tells us the new state s_1 that following a_1 from s_0 will lead to. We know that when s_0 is fixed, the target function on the second component a_1 should be an isomorphism $\mathfrak{s}[s_0] \to \mathfrak{s}(1)$; that is, there is exactly one direction at s_0 that leads to each state of $\mathfrak{s}$. But this property is a little tricky to state in the language of **Poly**; in fact, we won't attempt to do so just yet. Instead, we'll use it to make a notational choice: given $s, t \in \mathfrak{s}$, we will let $s \to t$ denote the unique direction at s that leads to t, so that $\mathrm{tgt}(s, s \to t) = t$. So we can redraw our transition lens as



(7.4)

In addition to the fact that $\mathrm{tgt}(s_0, s_0 \to s_1) = s_1$ as intended, this picture tells us two more properties of δ.

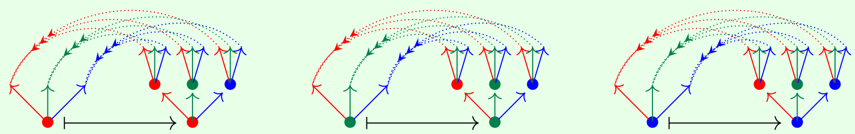
The first is that the bottom arrow is the identity on $\mathfrak{s}(1)$. This is something we would like to be able to express categorically in the language of **Poly**. We'll see that this property falls out naturally when we express how the transition lens plays nicely with the do-nothing section in Section 7.1.3.

The second is that the run arrow, which runs the transition $s_0 \to s_1$ and the transition $s_1 \to s_2$ together into a transition starting at s_0, should have the same target as tfhe second transition it follows: in this case, s_2. Equationally, writing the left and right hand sides only in terms of the contents of the blue boxes, we have that

$$\mathrm{tgt}(s_0, \mathrm{run}(s_0, s_0 \to s_1, s_1 \to s_2)) = s_2 = \mathrm{tgt}(\mathrm{tgt}(s_0, s_0 \to s_1), s_1 \to s_2). \qquad (7.5)$$

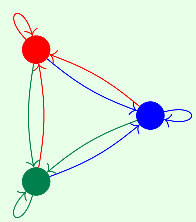
We'll see that this property arises naturally when we we generalize the transition lens to more than two steps in Section 7.1.4.

Example 7.6 (The transition lens in tree pictures). Continuing from Example 7.3, we draw the transition lens $\delta\colon \mathfrak{s} \to \mathfrak{s} \triangleleft \mathfrak{s}$ of $\mathfrak{s} := 3y^3 \cong \{\bullet, \bullet, \bullet\}y^{\{\bullet,\bullet,\bullet\}}$ (where directions are labeled with their targets) in tree pictures as well, recalling that the trees of $\mathfrak{s} \triangleleft \mathfrak{s}$ are obtained by taking an $\mathfrak{s}$-corolla and grafting more $\mathfrak{s}$-corollas to each of its leaves:



On positions, the target function of δ tells us which root of $\mathfrak{s}$ to graft onto each leaf of $\mathfrak{s}$. Then on directions, the run function of δ tells us how to collapse the height-2 leaves of the trees we obtain in $\mathfrak{s} \triangleleft \mathfrak{s}$ down to the original height-1 leaves of the corollas of $\mathfrak{s}$.

We can draw what the target function is doing more compactly by taking the corollas of $\mathfrak{s}$ and "bending the arrows" so that they point to their targets, like so:



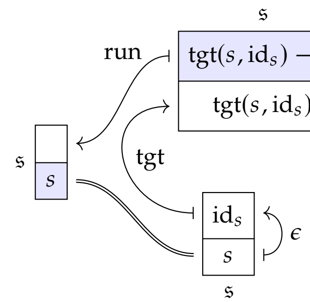
So the target function of δ turns our corolla picture of $\mathfrak{s}$ into a complete graph on its roots! Then the run function takes any two arrows that form a path in the graph and collapses them down to a single arrow that starts (and, according to (7.5), ends) at the same vertex as the two-arrow path.

If this all sounds suspiciously familiar to you, you're on the right track—hang tight.

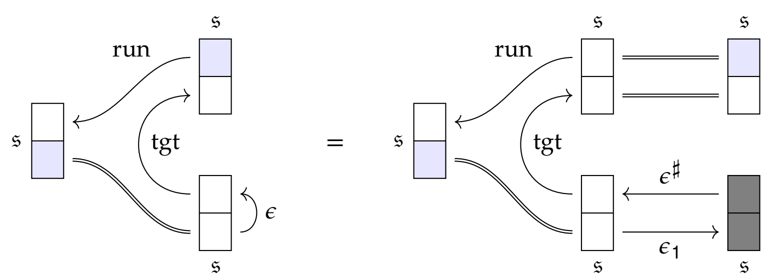
7.1.3 The do-nothing section coheres with the transition lens

For each state $s \in \mathfrak{s}(1)$, the do-nothing section $\epsilon\colon \mathfrak{s} \to y$ picks out the $\mathfrak{s}[s]$-direction id_s that "does nothing" and keeps the system in the same state s. But it is the transition lens $\delta\colon \mathfrak{s} \to \mathfrak{s} \triangleleft \mathfrak{s}$ that actually sets our state system in motion, specifying the target of each direction and how two directions run together. Either of these directions could be our do-nothing direction id_s, so let's try to figure out what should happen when we set each one in turn to id_s.

We can draw in polyboxes what happens when we set id_s, as specified by ϵ, to be the first direction that δ runs together like this:



Reading this picture from left to right, we see that it depicts the polyboxes of the composite lens $\delta \mathbin{⨟} (\epsilon \triangleleft \mathfrak{s})\colon \mathfrak{s} \to y \triangleleft \mathfrak{s} \cong \mathfrak{s}$ (recall that we sometimes denote the identity lens on $\mathfrak{s}$ also by $\mathfrak{s}$). To make this interpretation more transparent, we could be a little more verbose with our polybox picture if we wanted to (omitting the contents of the boxes for clarity):

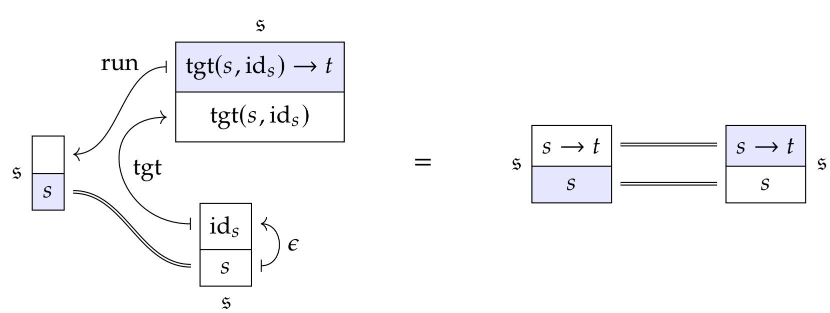


Now what should $\mathrm{tgt}(\mathrm{id}_s)$ be, and what should go in the direction box on the left?

If following the direction id_s from the state s is really the same as doing nothing, then its target state should be the same state s that it emerged from. Moreover, running together id_s with any other direction $s \to t$ from s should be no different from the direction $s \to t$ on its own. So

$$\mathrm{tgt}(s, \mathrm{id}_s) = s \qquad \text{and} \qquad \mathrm{run}(s, \mathrm{id}_s, s \to t) = s \to t.$$

In fact, id_s should really just be the direction $s \to s$. Pictorially, we have the equation

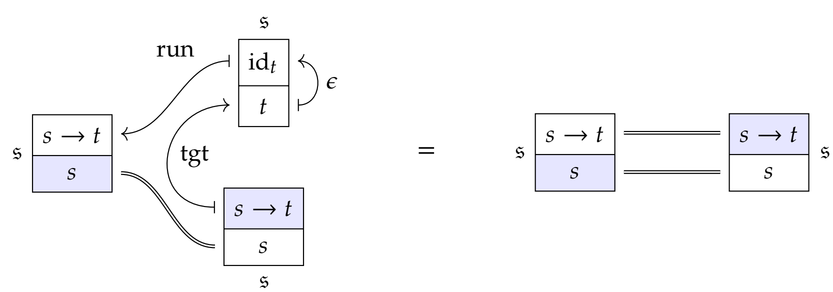


Or, if you prefer, we might say that $\delta \fatsemi (\epsilon \triangleleft \mathfrak{s}) = \mathrm{id}_{\mathfrak{s}}$, or that the following diagram commutes:

$$\begin{tikzcd}
y \triangleleft \mathfrak{s} \arrow[r, equal] & \mathfrak{s} \arrow[d, "\delta"] \\
& \mathfrak{s} \triangleleft \mathfrak{s} \arrow[ul, "\epsilon \triangleleft \mathfrak{s}"]
\end{tikzcd}$$

This commutative diagram captures one way in which ϵ and δ always relate—and it's written entirely in the language of **Poly**, without having to talk about individual sets!

What about setting the second direction that δ runs together to what is specified by ϵ, rather than the first? To answer this, we should look at the composite lens $\delta \fatsemi (\mathfrak{s} \triangleleft \epsilon) \colon \mathfrak{s} \to \mathfrak{s} \triangleleft y \cong \mathfrak{s}$ instead. But the do-nothing direction should still do nothing, so here's what the polybox picture should look like:



The lens depicted on the right hand side of the equation is again the identity lens on $\mathfrak{s}$.

If we match up the two white boxes on the right hand side of the equation with the corresponding white boxes on the left, we can actually read two equations off of this polybox picture. Matching up positions in the codomain tells us that the bottom arrow of δ on the left must send s to itself: it is the identity function on $\mathfrak{s}(1)$. Indeed, this is exactly what we wanted to say about that arrow in Section 7.1.2.

Meanwhile, matching up directions in the domain tells us that

$$\mathrm{run}(s, s \to t, \mathrm{id}_t) = s \to t,$$

as we would expect: id_t is just be the direction $t \to t$.

More concisely, we can express both these facts in **Poly** via the equation $\delta \fatsemi (\mathfrak{s} \triangleleft \epsilon) = \mathrm{id}_{\mathfrak{s}}$. The corresponding commutative diagram is as follows:

$$\begin{tikzcd}
\mathfrak{s} \arrow[r, equal] \arrow[d, "\delta"'] & \mathfrak{s} \triangleleft y \\
\mathfrak{s} \triangleleft \mathfrak{s}. \arrow[ur, "\mathfrak{s} \triangleleft \epsilon"'] &
\end{tikzcd}$$

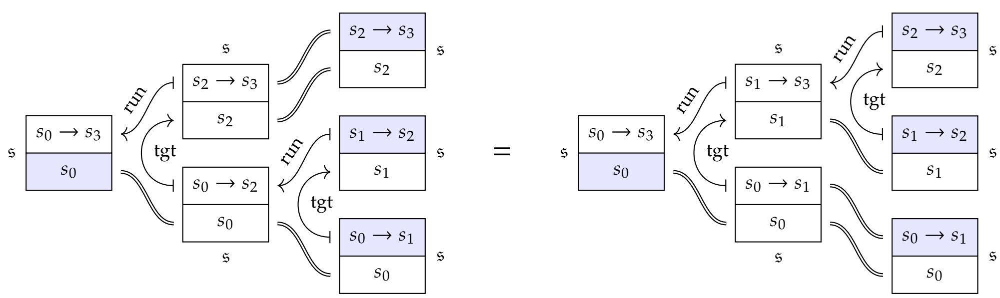
We can combine this with our previous commutative diagram to say that the relationship between the do-nothing section $\epsilon \colon \mathfrak{s} \to y$ and the transition lens $\delta \colon \mathfrak{s} \to \mathfrak{s} \triangleleft \mathfrak{s}$ of a state system $\mathfrak{s}$ is captured in **Poly** by the following commutative diagram:

$$
\begin{tikzcd}
y \triangleleft \mathfrak{s} \arrow[r, equal] & \mathfrak{s} \arrow[r, equal] \arrow[d, "\delta"] & \mathfrak{s} \triangleleft y \\
& \mathfrak{s} \triangleleft \mathfrak{s}. \arrow[ul, "\epsilon \triangleleft \mathfrak{s}"] \arrow[ur, "\mathfrak{s} \triangleleft \epsilon"'] &
\end{tikzcd}
\tag{7.7}
$$

7.1.4 The transition lens is coassociative

Toward the end of Example 6.44, we noted that while the transition lens $\delta\colon \mathfrak{s} \to \mathfrak{s} \triangleleft \mathfrak{s}$ gives us a canonical way to model two steps of a dynamical system with state system $\mathfrak{s}$, we have a choice of how to model three steps through the same system: we could obtain a lens $\mathfrak{s} \to \mathfrak{s}^{\triangleleft 3}$ that runs three directions together by taking either one of the composite lenses $\delta \mathbin{⨟} (\delta \triangleleft \mathfrak{s})$ or $\delta \mathbin{⨟} (\mathfrak{s} \triangleleft \delta)$. That presents a problem for us: which one should we choose?

Happily, it turns out this choice is a false one. If we write out the two composite lenses in polyboxes, with $\delta \mathbin{⨟} (\delta \triangleleft \mathfrak{s})$ on the left and $\delta \mathbin{⨟} (\mathfrak{s} \triangleleft \delta)$ on the right, we find that they are equal:



(7.8)

Remember: the way to read these polyboxes is to start at the lower blue square on the left and follow the path counter clockwise around the diagram; and if you reach a box with no arrows leading out of it, go up to the blue box above it and continue to follow the arrows from there.

There's a lot going on here, so let's break it down—we'll focus on the run functions first. On the left hand side, we run together $s_0 \to s_1$ and $s_1 \to s_2$ to obtain $s_0 \to s_2$, before running that together with $s_2 \to s_3$ to obtain $s_0 \to s_3$, as we see in the upper left box. Meanwhile, on the right, we run together $s_1 \to s_2$ and $s_2 \to s_3$ to obtain $s_1 \to s_3$, before running $s_0 \to s_1$ together with our newly obtained $s_1 \to s_3$ to again obtain $s_0 \to s_3$ in the upper left box. We could write this all out equationally, but all this is saying is that "running together" the directions of a state system is an associative operation. When running together three directions, it doesn't matter whether we run the first two together or the last two together to start. Not only is this guaranteed by the way in which we constructed δ, it also makes intuitive sense.

Exercise 7.9 (Solution here). Using only the contents of the blue boxes and the target and run functions, write down the equation that we can read off of (7.8) expressing the associativity of the "running together" operation. ◊

This associative property is what we get by matching up the white direction boxes on each domain side, but there are three more white position boxes on each codomain side that we can match up as well. The fact that the lower two of these pairs coincide is a consequence of the fact that the bottom arrow of δ is the identity, which we already knew from Section 7.1.3; so we don't learn anything new there. On the other hand, the fact that both the upper position boxes in the codomain contain s_2 implies that

$$\begin{aligned}\mathrm{tgt}(s_0, \mathrm{run}(s_0, s_0 \to s_1, s_1 \to s_2)) &= \mathrm{tgt}(s_0, s_0 \to s_2)\\ &= s_2\\ &= \mathrm{tgt}(s_1, s_1 \to s_2)\\ &= \mathrm{tgt}(\mathrm{tgt}(s_0, s_0 \to s_1), s_1 \to s_2),\end{aligned}$$

which is exactly what we wanted in (7.5). In English, this says that when we run together $s_0 \to s_1$ and $s_1 \to s_2$, the new direction's target is the same as the direction of $s_1 \to s_2$, the latter of the two directions that we ran together. Again, this coincides with our intuition: if we follow two directions in order, we should end up at wherever the latter direction leads us.

Hence both the associativity of running directions together and the relationship between the target and run functions from (7.5) are captured by the equality of lenses $\delta \mathbin{⨾} (\delta \triangleleft \mathfrak{s}) = \delta \mathbin{⨾} (\mathfrak{s} \triangleleft \delta)$. Equivalently, the following diagram in **Poly** commutes:

$$\begin{tikzcd} \mathfrak{s} \arrow[r, "\delta"] \arrow[d, "\delta"'] & \mathfrak{s} \triangleleft \mathfrak{s} \arrow[d, "\mathfrak{s} \triangleleft \delta"] \\ \mathfrak{s} \triangleleft \mathfrak{s} \arrow[r, "\delta \triangleleft \mathfrak{s}"'] & \mathfrak{s} \triangleleft \mathfrak{s} \triangleleft \mathfrak{s}. \end{tikzcd} \tag{7.10}$$

Another way to say this is that δ is *coassociative*: while δ is only a lens $\mathfrak{s} \to \mathfrak{s}^{\triangleleft 2}$ as defined, the commutativity of (7.10) tells us that the two ways of getting a lens $\mathfrak{s} \to \mathfrak{s}^{\triangleleft 3}$ out of δ are actually the same. (This is dual to an *associative* operation, which is a binary operation that gives rise to two identical ternary operations.)

So δ induces a canonical lens $\mathfrak{s} \to \mathfrak{s}^{\triangleleft 3}$, which we will call $\delta^{(3)}$, as it has 3 copies of $\mathfrak{s}$ in its codomain. Armed with this new lens, we can model three steps through a system $\varphi \colon \mathfrak{s} \to p$ with interface $p \in$ **Poly** as the composite lens

$$\mathfrak{s} \xrightarrow{\delta^{(3)}} \mathfrak{s}^{\triangleleft 3} \xrightarrow{\varphi^{\triangleleft 3}} p^{\triangleleft 3}.$$

In fact, coassociativity guarantees that δ induces a canonical lens $\delta^{(n)} \colon \mathfrak{s} \to \mathfrak{s}^{\triangleleft n}$ for every integer $n \geq 2$, starting with $\delta^{(2)} := \delta$.[1] For concreteness, we could then define

[1]Perhaps this notation seems a little unnatural, but it helps to think of the original $\delta \colon \mathfrak{s} \to \mathfrak{s} \triangleleft \mathfrak{s}$ as the $n = 2$ case of a generalized transition lens modeling n steps through the state system.

$\delta^{(n)}$ for $n > 2$ inductively by $\delta^{(n)} \coloneqq \delta \fatsemi (\delta^{(n-1)} \triangleleft \mathfrak{s})$, or just as well by $\delta^{(n)} \coloneqq \delta \fatsemi (\mathfrak{s} \triangleleft \delta^{(n-1)})$ or even $\delta^{(n)} \coloneqq \delta \fatsemi (\delta^{(\ell)} \triangleleft \delta^{(m)})$ for some pair of integers $\ell, m > 1$ satisfying $\ell + m = n$. Regardless, the coassociativity of δ means that it doesn't matter how we build a lens $\mathfrak{s} \to \mathfrak{s}^{\triangleleft(n+1)}$ out of $\delta, \fatsemi, \triangleleft$, and identity lenses: we'll always end up with the same lens. We will state this in more generality in Proposition 7.20, but here's some practice with the $n = 4$ case for a taste of what's to come.

Exercise 7.11 (Solution here).

1. Say we know nothing about $\mathfrak{s}$ or δ apart from the fact that $\mathfrak{s} \in$ **Poly** and that δ is a lens $\mathfrak{s} \to \mathfrak{s} \triangleleft \mathfrak{s}$. List all the ways to obtain a lens $\mathfrak{s} \to \mathfrak{s}^{\triangleleft 4}$ using only copies of $\delta, \mathrm{id}_{\mathfrak{s}}, \triangleleft$, and $\fatsemi$. (You may write $\mathfrak{s}$ for $\mathrm{id}_{\mathfrak{s}}$.)
2. Now assume that (7.10) commutes. Show that all the lenses on your list are equal. (Hint: Use the fact that $(f \fatsemi g) \triangleleft (h \fatsemi k) = (f \triangleleft h) \fatsemi (g \triangleleft k)$ for lenses f, g, h, k). ◊

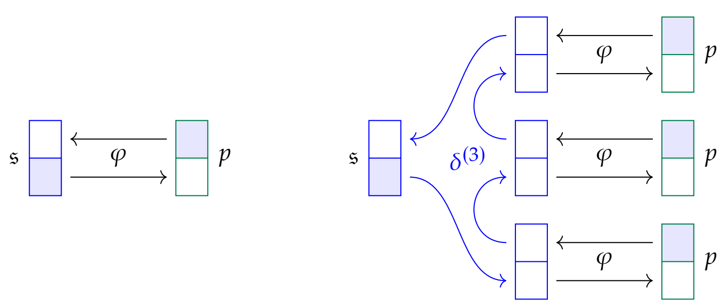
7.1.5 Running dynamical systems

Finally, we are ready to fulfill our promise from way back in Example 4.43 by using the language of **Poly** to describe stepping through a dynamical system n times for arbitrary $n \in \mathbb{N}$. Given a dynamical system $\varphi\colon \mathfrak{s} \to p$ with interface $p \in$ **Poly**, we can construct a new dynamical system that we call $\mathrm{Run}_n(\varphi)$, with the same state system $\mathfrak{s}$ but a new interface $p^{\triangleleft n}$, by defining $\mathrm{Run}_n(\varphi) \coloneqq \delta^{(n)} \fatsemi \varphi^{\triangleleft n}$. Visually, we define $\mathrm{Run}_n(\varphi)$ so that the following diagram commutes:

$$
\begin{tikzcd}
\mathfrak{s} \arrow[r, "\delta^{(n)}"] \arrow[rr, bend right, "\mathrm{Run}_n(\varphi)"'] & \mathfrak{s}^{\triangleleft n} \arrow[r, "\varphi^{\triangleleft n}"] & p^{\triangleleft n}
\end{tikzcd}
$$

One way to think of this is that $\mathrm{Run}_n(\varphi)$ is a sped-up version of φ: one step through $\mathrm{Run}_n(\varphi)$ is equivalent to n steps through φ. But this is just because a single interaction with the interface $p^{\triangleleft n}$ models a sequence of n interactions with the interface p, as detailed in Section 6.1.4 and Example 6.43. So $\mathrm{Run}_n(\varphi)$ repackages n cycles through φ into a single step. Crucially, $\delta^{(n)}$ is what tells us how to sequence all n of these steps together on the state system side. We illustrated how δ does this for the $n = 2$ case in Example 6.44, and here's a polybox picture for the $n = 3$ case:

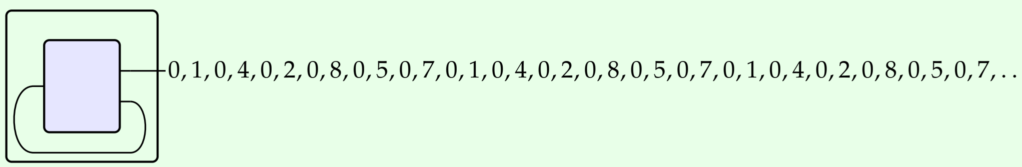
Given $\varphi\colon \mathfrak{s} \to p$, obtain $\mathrm{Run}_3(\varphi)$:



Notice that we have only defined $\delta^{(n)}$, and thus $\mathrm{Run}_n(\varphi)$, for integers $n \geq 2$. But $n = 0$ runs through n is doing nothing, modeled by the do-nothing section $\epsilon \colon \mathfrak{s} \to y$, while $n = 1$ run through φ is modeled by $\varphi \colon \mathfrak{s} \to \mathfrak{s}$ itself. So we want $\mathrm{Run}_0(\varphi) = \epsilon$ and $\mathrm{Run}_1(\varphi) = \varphi$; we can achieve this by setting $\delta^{(0)} := \epsilon$ and $\delta^{(1)} := \mathrm{id}_{\mathfrak{s}}$. Here we should think of the do-nothing section $\delta^{(0)}$ as the transition lens modeling 0 steps through our state system, and the identity $\delta^{(1)}$ as the transition lens modeling a single step.

Exercise 7.12 (Solution here). Verify that when $\delta^{(0)} = \epsilon$ and $\delta^{(1)} = \mathrm{id}_{\mathfrak{s}}$, if $\mathrm{Run}_n(\varphi)$ is defined as $\delta^{(n)} \mathbin{⨟} \varphi^{\triangleleft n}$ for all $n \in \mathbb{N}$, then $\mathrm{Run}_0(\varphi) = \epsilon$ and $\mathrm{Run}_1(\varphi) = \mathrm{id}_{\mathfrak{s}}$. ◊

Example 7.13 (Returning every other position). In Exercise 4.65, we built a dynamical system $\varphi \colon Sy^S \to \mathbb{N}y$ that returns natural numbers—specifically digits, alternating between 0's and the base-10 digits of 1/7 after the decimal point like so:



Say we only wanted the system to return the digits of 1/7 after the decimal point; we'd like to do away with all these 0's. In other words, we want a new system $Sy^S \to \mathbb{N}y$ that acts like φ, except that it only returns every other position that φ returns.

We could build such a system from scratch—or we can simply start from φ and apply Run_2, yielding a system $\mathrm{Run}_2(\varphi) \colon Sy^S \to \mathbb{N}y \triangleleft \mathbb{N}y \cong \mathbb{N}^2y$ that returns the positions of φ two at a time:

$$(0,1),(0,4),(0,2),(0,8),(0,5),(0,7),(0,1),(0,4),(0,2),(0,8),(0,5),(0,7),\ldots$$

Then we just need to compose $\mathrm{Run}_2(\varphi)$ with a lens $\pi_2 \colon \mathbb{N}^2y \to \mathbb{N}y$ equal to the second coordinate projection on positions (and the identity on directions) to extract the positions we want. The new system $Sy^S \to \mathbb{N}y$ that skips over every position of φ is therefore the following composite:

$$\begin{tikzcd} Sy^S \arrow[r, "\delta^{(2)}"] \arrow[rr, bend right, "\mathrm{Run}_2(\varphi)"'] & Sy^S \triangleleft Sy^S \arrow[r, "\varphi^{\triangleleft 2}"] & \mathbb{N}y \triangleleft \mathbb{N}y \cong \mathbb{N}^2y \arrow[r, "\pi_2"] & \mathbb{N}y. \end{tikzcd}$$

We can apply this technique in general to skip (or otherwise act on) the positions of a dynamical system at regular intervals.

One drawback of the $\mathrm{Run}_n(-)$ operation is that we need to keep track of a separate morphism $Sy^S \to p^{\triangleleft n}$ for every $n \in \mathbb{N}$, as well as various ways to relate these morphisms for different values of n. Is there a way to package all this information into a single morphism that can model arbitrarily long runs through the system? We will answer this question in Chapter 8; but for now, let us investigate what's really going on with our state systems algebraically.

7.1.6 State systems as comonoids

It turns out that objects equipped with morphisms like those in Sections 7.1.1 and 7.1.2 that satisfy the commutative diagrams from Sections 7.1.3 and 7.1.4 are well-known to category theorists.

Definition 7.14 (Comonoid)**.** In a monoidal category $(\mathbf{C}, y, \triangleleft)$, a *comonoid* $\mathscr{C} := (\mathfrak{c}, \epsilon, \delta)$ consists of

- an object $\mathfrak{c} \in \mathbf{C}$, called the *carrier*;
- a morphism $\epsilon \colon \mathfrak{c} \to y$ in $\mathbf{C}$, called the *eraser* (or the *counit*); and
- a morphism $\delta \colon \mathfrak{c} \to \mathfrak{c} \triangleleft \mathfrak{c}$ in $\mathbf{C}$, called the *duplicator* (or the *comultiplication*);

such that the following diagrams, collectively known as the *comonoid laws*, commute:

$$\begin{tikzcd}
y \triangleleft \mathfrak{c} \arrow[r, equal] & \mathfrak{c} \arrow[d, "\delta"] \arrow[r, equal] & \mathfrak{c} \triangleleft y \\
& \mathfrak{c} \triangleleft \mathfrak{c}, \arrow[ul, "\epsilon \triangleleft \mathfrak{c}"] \arrow[ur, "\mathfrak{c} \triangleleft \epsilon"'] &
\end{tikzcd} \tag{7.15}$$

where the left triangle is known as the *left erasure* (or *counit*) *law* and the right triangle is known as the *right erasure* (or *counit*) *law*; and

$$\begin{tikzcd}
\mathfrak{c} \arrow[r, "\delta"] \arrow[d, "\delta"'] & \mathfrak{c} \triangleleft \mathfrak{c} \arrow[d, "\mathfrak{c} \triangleleft \delta"] \\
\mathfrak{c} \triangleleft \mathfrak{c} \arrow[r, "\delta \triangleleft \mathfrak{c}"'] & \mathfrak{c} \triangleleft \mathfrak{c} \triangleleft \mathfrak{c},
\end{tikzcd} \tag{7.16}$$

known as the *coassociative law*.

We may also say that the eraser and duplicator morphisms comprise a *comonoid structure* on the carrier, or we may identify a comonoid with its carrier if the eraser and duplicator can be inferred from context.

We refer to a comonoid $\mathscr{C}$ in $(\mathbf{Poly}, y, \triangleleft)$ as a *polynomial comonoid*.

Remark 7.17. The concept of a *comonoid* in a monoidal category is dual to that of a *monoid*, which may be more familiar. Monoids come with *unit* and *multiplication* morphisms that point the other way, so named because they generalize the unit and multiplication operations of a monoid in **Set**. (We'll talk more about monoids in **Set** in Example 7.40.) Prepending 'co-' to each term yields the corresponding terms for comonoids.

The alternative names *eraser* for the *counit* and *duplicator* for the *comultiplication* are less standard, but we will favor them to avoid confusion between the counit of a *comonoid* and the counit of an *adjunction*—and so that their names match up with the Greek letters ϵ and δ that we will so often use to label them. The word "duplicator" comes from the fact that $\delta\colon \mathfrak{c} \to \mathfrak{c} \triangleleft \mathfrak{c}$ effectively turns one $\mathfrak{c}$ into two, while the "eraser" $\epsilon\colon \mathfrak{c} \to y$ erases the $\mathfrak{c}$ altogether, leaving only the monoidal unit y. Still, it can be helpful to think of comonoids as having a *coassociative* comultiplication along with a counit satisfying *left and right counit laws*.

Remark 7.18. Comonoids in a functor category with respect to the composition product are generally known as *comonads*. So it would be a little more precise and familiar to refer to our polynomial comonoids as *polynomial comonads*. But since we think of our polynomials more often in terms of positions and directions than as functors, we'll favor the term comonoid over comonad.

Example 7.19 (State systems are polynomial comonoids). Nearly all our work on state systems up until now can be summarized thusly:

every state system is a polynomial comonoid,
whose eraser is the do-nothing section
and whose duplicator is the transition lens.

The comonoid structure on a state system $\mathfrak{s}$ is what allows us to write canonical lenses $\mathfrak{s} \to \mathfrak{s}^{\triangleleft n}$ for any $n \in \mathbb{N}$. We can then model n steps through a dynamical system $\varphi\colon \mathfrak{s} \to p$ with interface $p \in \mathbf{Poly}$ by composing this canonical lens with $\varphi^{\triangleleft n}$ to obtain a "sped-up" dynamical system $\mathrm{Run}_n(\varphi)$. This new system has the same state system $\mathfrak{s}$, but its interface is now $p^{\triangleleft n}$.

The canonicity of $\mathfrak{s} \to \mathfrak{s}^{\triangleleft n}$ is due to the following standard result about comonoids, which can be proved inductively.

Proposition 7.20 (Defining $\delta^{(n)}$)**.** Given a comonoid $(\mathfrak{c}, \epsilon, \delta)$, let $\delta^{(n)}\colon \mathfrak{c} \to \mathfrak{c}^{\triangleleft n}$ be given as follows. Let $\delta^{(0)} := \epsilon$ and inductively define $\delta^{(n+1)} := \delta \mathbin{⨾} \left(\delta^{(n)} \triangleleft \mathfrak{c}\right)$ for all $n \in \mathbb{N}$. Then we have the following:

(a) $\delta^{(n)}$ is a morphism $\mathfrak{c} \to \mathfrak{c}^{\triangleleft n}$ for all $n \in \mathbb{N}$;
(b) $\delta^{(1)} = \mathfrak{c} = \mathrm{id}_{\mathfrak{c}}$;
(c) $\delta^{(2)} = \delta$; and
(d) $\delta^{(n)} = \delta \mathbin{⨾} \left(\delta^{(k)} \triangleleft \delta^{(n-k)}\right)$ for all $k, n \in \mathbb{N}$ with $k \leq n$, so our choice of morphism $\mathfrak{c} \to \mathfrak{c}^{\triangleleft(n+1)}$ is canonical.

Proof. We leave parts (a), (b), and (c) for Exercise 7.21. Part (d) amounts to coassociativity. □

We'll continue to use the notation introduced here throughout for general comonoids.

Exercise 7.21 (Solution here). Prove the first three parts of Proposition 7.20.

1. Prove part (a).
2. Prove part (b).
3. Prove part (c). ◊

Example 7.22 (Not all polynomial comonoids are state systems). At this point, a natural question to ask is whether everything we know about a state system $\mathfrak{s}$ is captured by the fact that state systems are polynomial comonoids. In other words, are state systems the only polynomial comonoids there are?

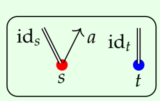
The answer turns out to be no. After all, there is one fact about state systems from Section 7.1.2 that we did not encode in **Poly**: for a fixed state $s \in \mathfrak{s}(1)$, the target function $\mathfrak{s}[s] \to \mathfrak{s}(1)$ sending directions at s to their target states is a bijection.

Nothing in our comonoid laws guarantees this bijectivity. An arbitrary polynomial comonoid might send different directions at s to the same target—given a second state t, there may be multiple ways to get from s to t. It might even send *no* directions at s to a target t, making it impossible to get from s to t. (We'll give an explicit example of a comonoid that is not a state system in Example 7.23.) State systems as we have defined them are just the polynomial comonoids that do not allow either of these variations, for which the bijective property holds.

We consider this a feature, not a bug. After all, it is an abstraction to say that there is exactly one way to get from any one state in a system to another. It is perfectly plausible that the inner workings of a state system do not permit traveling between some states and differentiate ways of traveling between others. We won't formally introduce this idea into our theory of dependent dynamical systems, but we will often think of polynomial comonoids as a sort of generalized state system throughout the rest of the book.

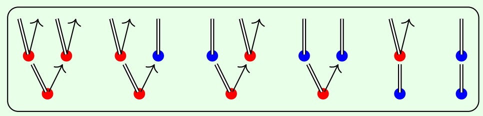
Example 7.23 (A comonoid that is not a state system). The polynomial $y^2 + y$ is not a state system: one of its direction-sets has one fewer element than its position-set. But it can still be given a comonoid structure. We describe that structure here, but we will go a little quickly, because we'll soon discover a much more familiar way to think about comonoids.

Define $\mathfrak{a} := \{s\}y^{\{\mathrm{id}_s, a\}} + \{t\}y^{\{\mathrm{id}_t\}} \cong y^2 + y$. Here is its tree picture:

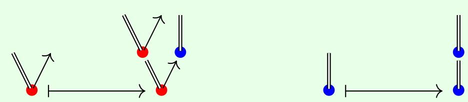
$\mathfrak{a} :=$ 

Notice that we have drawn one direction out of each position—id_s and id_t—with a double bar. We let these be the directions that the eraser $\epsilon\colon \mathfrak{a} \to y$ picks out. The double bar is meant to evoke an equals sign from the root position to the eventual target position, which is appropriate, as these two positions should be equal for every direction that the eraser selects. We can draw the selections that ϵ makes like so:

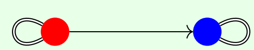
Now we need a duplicator $\delta\colon \mathfrak{a} \to \mathfrak{a} \triangleleft \mathfrak{a}$. Before we define it, let's draw out $\mathfrak{a} \triangleleft \mathfrak{a}$ to see what it looks like. Remember that we need to graft corollas of $\mathfrak{a}$ onto leaves of $\mathfrak{a}$ in every possible way:

$\mathfrak{a} \triangleleft \mathfrak{a} =$ 

Each of these trees gives a way to match directions out of one position to positions they could lead to. On positions, δ will decide which matchings to pick by sending the red s to one of the four positions on the left and the blue t to one of the two positions on the right. We want the double-barred directions that the eraser picked out to have the same position on either end (in fact, the erasure laws guarantee this). So the only choice to be made is whether we want the other direction a at s to point to s or to t. Let's pick t for the time being, so that on positions, δ looks like this:

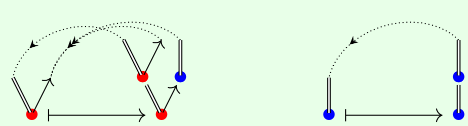


As in Example 7.6, we can interpret this as telling us how to "bend" the arrows of $\mathfrak{a}$ so that they point to other positions:



(7.24)

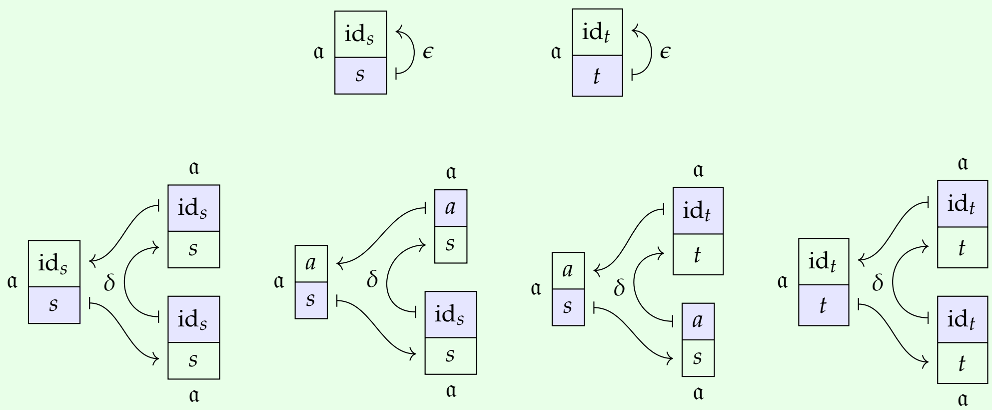
Meanwhile, on directions, δ should tell us how to run two directions together into one. Fortunately, there's not much for us to do here—we know that if one of the two directions δ runs together is one of the double-barred directions that the eraser picked out, then δ should ignore that "do-nothing" direction and yield the other direction (again, the erasure laws ensure this). Here's what that looks like:



(7.25)

And that's all we need to specify the triple $(\mathfrak{a}, \epsilon, \delta)$.

Here are $\epsilon \colon \mathfrak{a} \to y$ and $\delta \colon \mathfrak{a} \to \mathfrak{a} \triangleleft \mathfrak{a}$ again, in terms of polyboxes.



Of course, we have yet to check that $(\mathfrak{a}, \epsilon, \delta)$ really is a comonoid, i.e. that the diagrams in (7.15) and (7.16) commute. We leave that for Exercise 7.26.

Exercise 7.26 (Solution here). Verify that $(\mathfrak{a}, \epsilon, \delta)$ as defined in Example 7.23 obeys the erasure laws in (7.15) and the coassociative law in (7.16). ◊

Exercise 7.27 (Solution here). Show that if B is a set, then there exists a unique comonoid structure on the linear polynomial By. ◊

Once you know that state systems are comonoids in **Poly**, but not the only ones, the natural question to ask is "what are all the other comonoids in **Poly**?" Or perhaps, as we led you through this case study of $\mathfrak{s}$, you have already suspected the truth: a polynomial comonoid—what with its directions leading from one position to another, directions that can be run together associatively among which there are directions at every position that do nothing—is just another name for a category.

7.2 Polynomial comonoids are categories

What Ahman and Uustalu showed was that polynomial comonoids 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 their revelation to be truly shocking, and it suggests some very different ways to think about categories. But let's go over their result first.

Theorem 7.28 (Ahman-Uustalu)**.** There is a one-to-one isomorphism-preserving correspondence between polynomial comonoids and (small) categories.

Our goal is to spell out this correspondence so that we can justly proclaim:

Comonoids in **Poly** *are precisely categories!*

7.2.1 Translating between polynomial comonoids and categories

First, we describe how to translate between the carrier $\mathfrak{c}$ of a comonoid $\mathscr{C} := (\mathfrak{c}, \delta, \epsilon)$ and the objects and morphisms of the corresponding category $\mathcal{C}$. The idea is pretty simple, and you may have already guessed it: positions are objects and directions are morphisms.

Positions as objects, directions as morphisms

More precisely, the positions of $\mathfrak{c}$ are the objects of $\mathcal{C}$:

$$\mathfrak{c}(1) = \operatorname{Ob} \mathcal{C}. \tag{7.29}$$

Then for each such position or object i, the $\mathfrak{c}[i]$-directions are the morphisms of $\mathcal{C}$ with domain i:

$$\mathfrak{c}[i] = \sum_{j \in \operatorname{Ob} \mathcal{C}} \mathcal{C}(i, j). \tag{7.30}$$

The right hand side above is a little clumsier than the left; this is because while we are used to thinking of hom-sets of categories such as $\mathcal{C}(i, j)$, consisting of all morphisms in $\mathcal{C}$ with a fixed domain and codomain, we aren't used to thinking about the collection of all morphisms in $\mathcal{C}$ with a fixed domain and an arbitrary codomain quite as often.[2] On the other hand, the carrier *only* encodes which morphisms have each object as its domain, i.e. which directions are at each position. Codomains will be encoded in the data of the comonoid elsewhere.

This is the key difference in perspective between the polynomial comonoid perspective of categories, in contrast to our usual hom-set perspective: the polynomial perspective is in a sense domain-centric, as highlighted by the following definition.

Definition 7.31 (Polynomial carrier)**.** Let $\mathcal{C}$ be a category. For every object i in $\mathcal{C}$, denote the morphisms in $\mathcal{C}$ with domain i by $\mathcal{C}[i]$, so that[3]

$$\mathcal{C}[i] := \sum_{j \in \operatorname{Ob} \mathcal{C}} \mathcal{C}(i, j).$$

Then the *polynomial carrier*, or simply *carrier*, of $\mathcal{C}$ is the polynomial

$$\sum_{i \in \operatorname{Ob} \mathcal{C}} y^{\mathcal{C}[i]}.$$

[2] Except, perhaps, in the context of coslice categories.

So everything we have said so far about the correspondence from Theorem 7.28 can be summarized by saying that it preserves carriers: the carrier of the category $\mathcal{C}$ is the carrier $\mathfrak{c}$ of the comonoid $\mathscr{C}$, so that $\operatorname{Ob}\mathcal{C} = \mathfrak{c}(1)$ and $\mathcal{C}[i] = \mathfrak{c}[i]$.

Remark 7.32. If we take the perspective that categories are equal if and only if their objects and morphisms are equal and obey the same laws, and similarly that polynomials are equal if and only if their position-sets and direction-sets are equal sets, then (7.29) and (7.30) really can be just strict equalities. This is why we are comfortable naming a "one-to-one correspondence" in Theorem 7.28 rather than just, say, some form of equivalence. Since the positions and directions of our polynomials always form *sets*, however, the categories we obtain under this correspondence are also necessarily *small*: their objects form a set, as do all of their morphisms. But we won't worry too much about size issues beyond this.

Exercise 7.33 (Solution here). What is the carrier of each of the following categories (up to isomorphism)?

1. The category

$$\boxed{A \xrightarrow{f} B}$$

where we have drawn every morphism except for the identity morphisms.
2. The category

$$\boxed{B \xrightarrow{g} A \xleftarrow{h} C}$$

where we have drawn every morphism except for the identity morphisms.
3. The empty category.
4. A category with exactly 1 object and a morphism i, for which every morphism can be written uniquely as the n-fold composite of i for some $n \in \mathbb{N}$.
5. The category

$$\boxed{0 \to 1 \to 2 \to 3 \to \cdots}$$

where there is a unique morphism $m \to n$ if $m \leq n$ (and no other morphisms).
6. The category

$$\boxed{0 \leftarrow 1 \leftarrow 2 \leftarrow 3 \leftarrow \cdots}$$

where there is a unique morphism $m \leftarrow n$ if $m \leq n$ (and no other morphisms). ◊

But a category $\mathcal{C}$ is more than its carrier polynomial, just as a comonoid $\mathscr{C}$ is more than its carrier $\mathfrak{c}$. In particular, we have said nothing about the codomains of morphisms in $\mathcal{C}$, nor anything about identity morphisms, composition, or how the

[3] We may also write $f : i \to \_$ to denote an arbitrary morphism $f \in \mathcal{C}[i]$, i.e. a morphism f in $\mathcal{C}$ with domain i and an unspecified codomain.

laws of a category are satisfied. Similarly, we have said nothing about the eraser ϵ or the duplicator δ of $\mathscr{C}$, nor anything about how the comonoid laws are satisfied. It turns out that all of these constituents and laws correspond to one another, as summarized by the following table. Here each item in the comonoid column—either a polynomial, a lens, or a lens equation—spans two rows, with the top row corresponding to positions and the bottom row corresponding to directions.

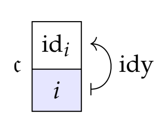
Comonoid	$\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$	**Category**	$\mathcal{C}$
carrier	$i \in \mathfrak{c}(1)$	objects	$i \in \operatorname{Ob}\mathcal{C}$
	$f \in \mathfrak{c}[i]$	morphisms	$f\colon i \to \_$
eraser	$\epsilon_1\colon \mathfrak{c}(1) \to 1$	—	—
	$\epsilon_i^\sharp\colon 1 \to \mathfrak{c}[i]$	identities	$\mathrm{id}_i\colon i \to \_$
duplicator	$\delta_1\colon \mathfrak{c}(1) \to (\mathfrak{c} \triangleleft \mathfrak{c})(1)$	codomains*	$\operatorname{cod}\colon \mathcal{C}[i] \to \operatorname{Ob}\mathcal{C}$
	$\delta_i^\sharp\colon (\mathfrak{c} \triangleleft \mathfrak{c})[\delta_1(i)] \to \mathfrak{c}[i]$	composition*	$⨟$
right erasure law		*	
		right identity law	
left erasure law		codomains of identities	$\operatorname{cod}\mathrm{id}_i = i$
		left identity law	
coassociative law		codomains of composites	$\operatorname{cod}(f ⨟ g) = \operatorname{cod} g$
		associative law of composition	

Note that the on-positions function of ϵ, being a function into the terminal set, encodes no actual data. The asterisk $*$ indicates that the right erasure law on positions works together with the duplicator to ensure that codomains and composites are properly specified.

We have already covered the correspondence between the first two rows, so let us consider each of the following rows in turn. In some sense, we have already seen each piece of this correspondence in action for state systems in Section 7.1, so we'll go through it a little faster this time for the general case.

The eraser assigns identities

We know that the eraser $\epsilon\colon \mathfrak{c} \to y$ can be identified with a dependent function $(i \in \mathfrak{c}(1)) \to [i]$, sending each position $i \in (1)$ to a $[i]$-direction. In terms of our category $\mathcal{C}$, the eraser sends each object $i \in \operatorname{Ob}\mathcal{C}$ to a morphism $i \to \_$. But this is exactly what we need to specify identity morphisms—a morphism out of each object. So the eraser of $\mathfrak{c}$ specifies the identity morphisms of the corresponding category $\mathcal{C}$. We can interpret the polybox picture for ϵ like so:

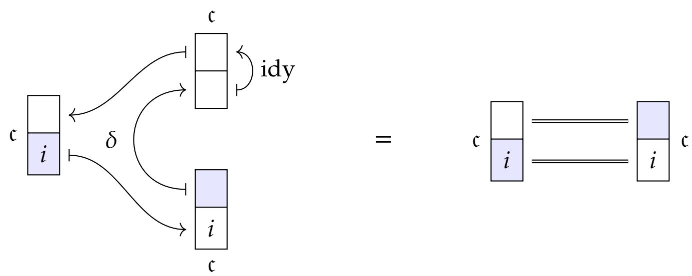


Here we have given the label idy to the arrow sending objects to their identity morphisms.

Keep in mind that from the domain-centric polynomial perspective, we have not yet specified that the codomain of an identity morphism is equal to its domain; that comes later.

The right erasure law on positions: a bit of bookkeeping

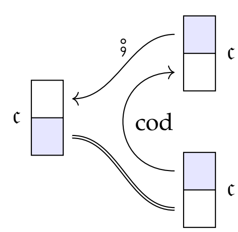
Keeping our label idy for the arrow in ϵ, the right erasure law $\delta \fatsemi (\mathfrak{c} \triangleleft \epsilon) = \mathrm{id}_{\mathfrak{c}}$ from (7.15) can be drawn in polyboxes like so:



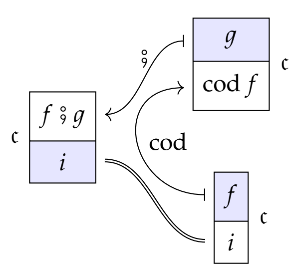
We have only filled in a few of the boxes, but that is enough to interpret what the right erasure law tells us on positions: that the bottom arrow of the duplicator must be the identity function on $\mathfrak{c}(1)$. Equipped with this knowledge, we can focus our attention on the other two arrows of δ.

The duplicator assigns codomains and composites

In fact, in the polybox picture for $\delta \colon \mathfrak{c} \to \mathfrak{c} \triangleleft \mathfrak{c}$, the middle arrow specifies codomains, and the top arrow specifies composition. We therefore label these arrows as follows:[4]



To check that this makes sense, we fill in the boxes:

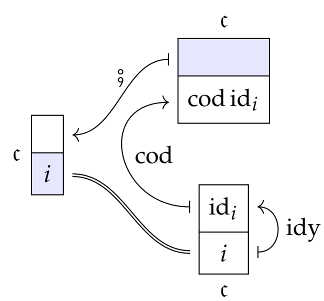


[4]Compare these labels to the names "target" and "run" that we gave to the arrows of a state system's transition lens.

Remember: each position box contains an object of $\mathcal{C}$, while each direction box contains a morphism of $\mathcal{C}$ emanating from the object below. So δ takes an object $i \in \mathrm{Ob}\,\mathcal{C}$ and a morphism $f\colon i \to \_$ in $\mathcal{C}$ and assigns another object $\operatorname{cod} f \in \mathrm{Ob}\,\mathcal{C}$ to be the codomain of f. It then takes another morphism $g\colon \operatorname{cod} f \to \_$ in $\mathcal{C}$ and assigns a morphism $f \mathbin{⨟} g\colon i \to \_$ to be the composite of f and g. In this way, every morphism gets a codomain, and every pair of morphisms that can be composed (i.e. the codomain of one matches the domain of the other) is assigned a composite. As with the identity morphism, we don't know what the codomain of this composite morphism is yet; but we do know that the domain of $f \mathbin{⨟} g$ matches the domain of f, as it should.

The left erasure law on positions: codomains of identities

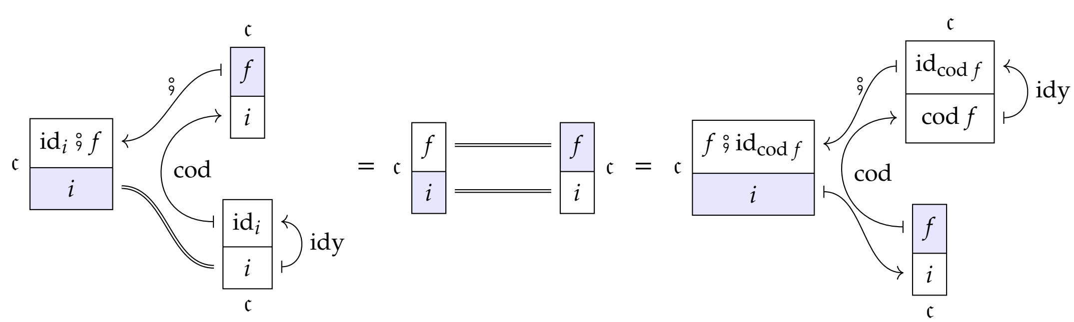
As with the right erasure law, we can partially fill in the polyboxes for the left erasure law $\delta \mathbin{⨟} (\epsilon \triangleleft \mathfrak{c}) = \mathrm{id}_{\mathfrak{c}}$ from (7.15) to read what it says on positions:



So the left erasure law on positions guarantees that $\operatorname{cod} \mathrm{id}_i = i$ for all $i \in \mathrm{Ob}\,\mathcal{C}$. It makes sense that we would find this here: the eraser assigns identities, while the duplicator assigns codomains, so a statement about codomains of identities is a coherence condition between the eraser and the duplicator.

The erasure laws on directions are the identity laws

Let us finish filling in the polyboxes for the left and right erasure laws to see what they have to say on directions. In the picture below, the left equality depicts the left erasure law (to conserve space, we'll substitute i for $\operatorname{cod} \mathrm{id}_i$ on the left, which we now know we can do), while the right equality depicts the right erasure law:



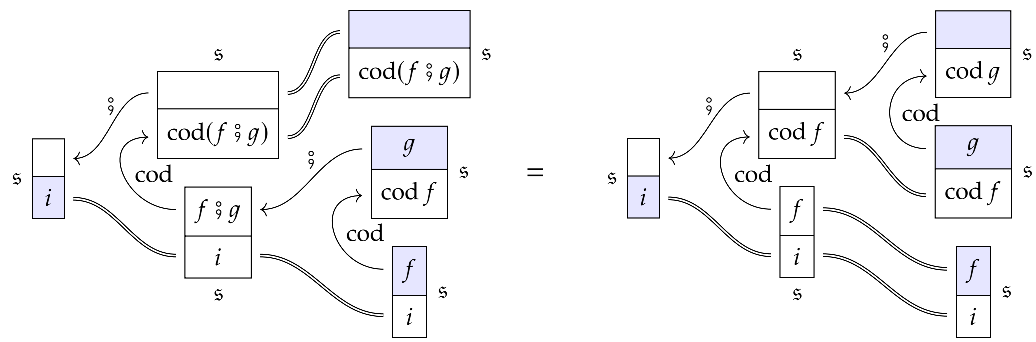
We find that on directions, the erasure laws state that for every object $i \in \operatorname{Ob}\mathcal{C}$ and morphism $f\colon i \to \_$ in $\mathcal{C}$,

$$\mathrm{id}_i \mathbin{⨾} f = f = f \mathbin{⨾} \mathrm{id}_{\operatorname{cod} f}.$$

But these are precisely the identity laws of the category $\mathcal{C}$.

The coassociative law on positions: codomains of composites

It remains to consider the comonoid's coassociative law (7.16), $\delta \mathbin{⨾} (\delta \triangleleft \mathfrak{s}) = \delta \mathbin{⨾} (\mathfrak{s} \triangleleft \delta)$. To read what it says on positions, we draw the polyboxes and fill them in, stopping just short of the uppermost direction box of the codomain:



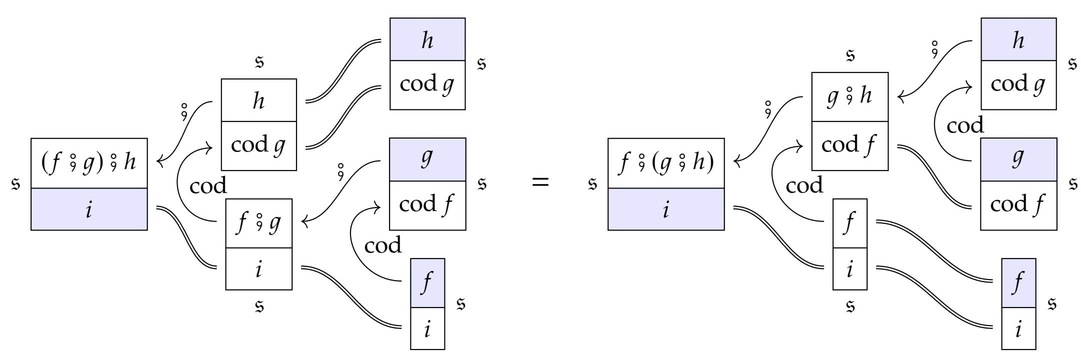
So on positions, the coassociative law states that given an object $i \in \operatorname{Ob}\mathcal{C}$ and morphisms $f\colon i \to \_$ and $g\colon \operatorname{cod} f \to \_$ in $\mathcal{C}$,

$$\operatorname{cod}(f \mathbin{⨾} g) = \operatorname{cod} g.$$

Hence composites are assigned the proper codomains.

The coassociative law on directions is the associative law of composition

Finally, let us fill in the remaining polyboxes for the coassociative law (we'll substitute $\operatorname{cod} g$ for $\operatorname{cod}(f \mathbin{⨾} g)$ on the left, which we now know we can do):



Thus, on directions, the coassociative law states that given an object $i \in \operatorname{Ob}\mathcal{C}$ and morphisms $f\colon i \to \_$, $g\colon \operatorname{cod} f \to \_$, and $h\colon \operatorname{cod} g \to \_$ in $\mathcal{C}$,

$$(f \mathbin{⨾} g) \mathbin{⨾} h = f \mathbin{⨾} (g \mathbin{⨾} h).$$

But this is precisely the associative law of composition in a category.

We've seen that the data and equations of polynomial comonoids correspond exactly to the data and equations of categories. This proves Theorem 7.28.

Generalized duplicators as unbiased composition

Before we move onto examples, one more note about the theory: notice that both sides of our coassociative law are given by $\delta^{(3)}: \mathfrak{c} \to \mathfrak{c}^{\triangleleft 3}$, as defined in Proposition 7.20. On directions, $\delta^{(3)}$ tells us how to compose three morphisms $i \xrightarrow{f} \_ \xrightarrow{g} \_ \xrightarrow{h} \_$ in $\mathcal{C}$ all at once to obtain $i \xrightarrow{f \fatsemi g \fatsemi h} \_$, and (co)associativity ensures this is well-defined.

In general, $\delta^{(n)}: \mathfrak{c} \to \mathfrak{c}^{\triangleleft n}$ on directions tells us how to compose n morphisms in $\mathcal{C}$ for each $n \in \mathbb{N}$. After all, we have already seen that $\delta^{(2)} = \delta$ performs binary composition, that $\delta^{(1)} = \mathrm{id}_{\mathfrak{c}}$ performs "unary" composition (the "unary composite" of a single morphism f is just f itself), and that $\delta^{(0)} = \epsilon$ performs "nullary" composition (the "nullary composite" at any object is just its identity). The directions of $\mathfrak{c}^{\triangleleft n}$ at positions in the image of $\delta^{(n)}$ are exactly the sequences of composable morphisms of length n, and $\delta^{(n)}$ sends each sequence to the single direction that is its composite.

7.2.2 Examples of categories as comonoids

Now that we know that polynomial comonoids are just categories, let's review some simple examples of categories and see how they may be interpreted as comonoids. As we go through these examples, pay attention to how the polynomial perspective causes us to view these familiar categories somewhat differently than usual.

Preorders

A *preorder* (or *thin category*) is a category in which every morphism $f: c \to d$ is the *only* morphism $c \to d$.[5] Composition in preorders is easy to describe, because the composite of $c \to d$ and $d \to e$ is always just the unique arrow $c \to e$. As such, preorders are some of the simplest examples of categories to consider—we already saw several in Exercise 7.33—so let us interpret these as comonoids first.

Example 7.34. Let us revisit Example 7.23, where we first wrote down a comonoid that was not a state system. We defined $\mathfrak{a} := \{s\}y^{\{\mathrm{id}_s, a\}} + \{t\}y^{\{\mathrm{id}_t\}} \cong y^2 + y$ and gave it a comonoid structure, with eraser $\epsilon: \mathfrak{a} \to y$ specifying directions id_s and id_t and duplicator $\delta: \mathfrak{a} \to \mathfrak{a} \triangleleft \mathfrak{a}$ pointing the direction a at t.

Looking at the picture we drew of the comonoid in (7.24), it should come as no surprise that the corresponding category $\mathcal{A}$ is the *walking arrow category*, which is a

[5] Sometimes these are also called *posets*, short for *partially ordered sets*, but strictly speaking the only isomorphisms in a poset are its identities, while a preorder allows objects to be isomorphic without being equal.

preorder with two objects and one morphism between them:

$$\mathcal{A} := \boxed{s \xrightarrow{a} t}$$

Here we omit the identity morphisms from our picture, but we know that they exist.

The category $\mathcal{A}$ has two objects, the $\mathfrak{a}$-positions s and t. It has two morphisms with domain s, the $\mathfrak{a}[s]$-directions id_s and a; and one morphism with domain t, the $\mathfrak{a}[t]$-direction id_t. The morphisms id_s and id_t picked out by the erasure are the identity morphisms, and the duplicator assigns them codomains that are equal to their domains. The duplicator also assigns a the codomain t; and as $\mathcal{A}$ is then a preorder, composites are determined automatically.

Exercise 7.35 (Solution here). Let $(\mathfrak{c}, \epsilon, \delta)$ be the comonoid corresponding to the preorder depicted as follows (identity morphisms omitted):

$$\boxed{B \xleftarrow{f} A \xrightarrow{g} C}$$

1. What is the carrier $\mathfrak{c}$?
2. Characterize the eraser ϵ.
3. Characterize the duplicator δ. ◊

Exercise 7.36 (Solution here). We showed in Exercise 7.27 that for any set B, the linear polynomial By has a unique comonoid structure. To what category does this comonoid correspond? ◊

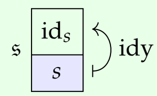
Exercise 7.37 (Solution here).

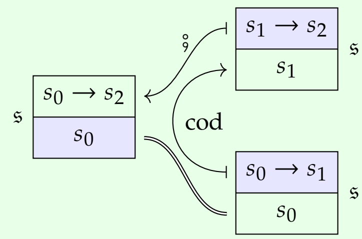
1. Find a comonoid structure for the polynomial $p := \mathsf{y}^{\mathsf{n}+1} + \mathsf{n}y$ whose corresponding category is a preorder. (It is enough to fully describe the category that it corresponds to.)
2. Would you call your category "star-shaped"? ◊

Example 7.38 (State systems as categories). We know that every state system $\mathfrak{s} \cong Sy^S$ with its do-nothing section $\epsilon\colon \mathfrak{s} \to y$ and its transition lens $\delta\colon \mathfrak{s} \to \mathfrak{s} \triangleleft \mathfrak{s}$ is a comonoid, so what category $\mathcal{S}$ does $(\mathfrak{s}, \epsilon, \delta)$ correspond to?

Recall from Example 7.22 that state systems are exactly those comonoids whose codomain (i.e. "target") functions $\mathrm{cod}\colon \mathfrak{s}[s] \to \mathfrak{s}(1)$ for $s \in \mathfrak{s}(1)$ are bijections. That is, from every object $s \in \mathrm{Ob}\,\mathcal{S} = \mathfrak{s}(1)$, there is exactly 1 morphism to every object $t \in \mathrm{Ob}\,\mathcal{S}$. So not only is $\mathcal{S}$ a preorder, it is the *codiscrete preorder* on $\mathfrak{s}(1)$, where there is always a morphism between every pair of objects.

Let's redraw the polyboxes for the do-nothing section of $\mathfrak{s}$ from (7.1) and the transition lens of $\mathfrak{s}$ from (7.4), this time with our new arrow labels, as a sanity check:

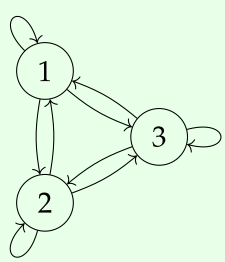


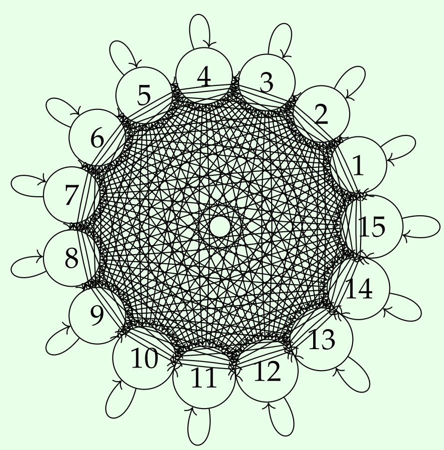


Indeed, we had already been writing the directions of $\mathfrak{s}$ as arrows $s \to t$, knowing that each was uniquely specified by its source s and its target t in $\mathfrak{s}(1)$. And in Section 7.1.3, we had already noted that id_s was just the arrow $s \to s$. So state systems have been categories with exactly one morphism between every pair of objects all along.

Other names for this category include the *indiscrete preorder* and the *codiscrete* or *indiscrete category*. These names highlight the fact that every object of this category is isomorphic to every other object: in fact, every arrow $s \to t$ is an isomorphism with inverse $t \to s$, for these compose as $\mathrm{id}_s \colon s \to s$ in one direction and $\mathrm{id}_t \colon t \to t$ in the other. Thus this category is also a *groupoid*, and it may be called the *codiscrete*, *indiscrete*, or *contractible groupoid*... but we will call it the *state category on S*, where S is the set of positions of $\mathfrak{s}$ or objects of $\mathcal{S}$.

Here are the state categories on $\mathsf{3}$ and on $\mathsf{15}$, with all maps (even identities) drawn:





The picture on the left should look familiar: it's what we drew in Example 7.6 when took the corolla picture for $\mathsf{3}y^{\mathsf{3}}$ and bent the arrows to point at their targets according to its transition lens. Notice that the graphs we obtain in this way are always complete.

Exercise 7.39 (Solution here). Let S be a set. Is there any comonoid structure on Sy^S other than that of the state category? ◊

Not only does Example 7.38 finally explain what our state systems really are (they're just special categories!), it illustrates two important features of our story. One is that on positions, the duplicator $\delta\colon \mathfrak{c} \to \mathfrak{c} \triangleleft \mathfrak{c}$ of a comonoid takes the corolla picture of $\mathfrak{c}$ and "bends the arrows" so that they point to other roots, yielding the underlying graph of a category. Then δ on directions collapses two-arrow paths in the graph down to individual arrows, while the eraser $\epsilon\colon \mathfrak{c} \to y$ identifies empty paths with identity arrows.

Another important point is that we can view any category as a *generalized state system*: its objects as *states*, and its morphisms as *transitions* between states. The polynomial comonoid perspective is particularly suited for thinking about categories in this way: each object is a position that we could be in, and each morphism out of that object is a direction that we might take. What is special about a comonoid is that each direction will always have another position at the end of it, making it reasonable to think of these directions as transitions between different states; and any sequence of transitions that we can follow is itself a transition.

Comparing these ideas, we see that they say the same thing: the first from the perspective of trees and graphs, the second from the perspective of dynamics. We might say that

a comonoid structure on a corolla forest turns
roots into vertices and
leaves into composable arrows between vertices;

or that

a comonoid structure on an polynomial turns
positions into states and
directions into composable transitions between states.

Monoids and monoid actions

Here we use *monoid* to refer to a monoid in the monoidal category $(\mathbf{Set}, 1, \times)$. We denote such a monoid by $(M, e, *)$, where M is the underlying set, $e \in M$ is the unit, and $*\colon M \times M \to M$ is the binary operation.

Example 7.40 (Monoids as representable comonoids). Recall that every monoid $(M, e, *)$ can be identified with a 1-object category $\mathcal{M}$ with a single hom-set M, a single identity morphism e, and composition given by $*$. Now we know that a 1-object category $\mathcal{M}$ is also a polynomial comonoid $(\mathfrak{m}, \epsilon, \delta)$ whose carrier has 1 position, with all of the morphisms of $\mathcal{M}$ becoming its directions. So the carrier of $\mathcal{M}$ is the representable polynomial y^M.

Then the eraser $\epsilon\colon y^M \to y$ picks out the identity morphism $e \in M$ on directions, while the duplicator $\delta\colon y^M \to y^M \triangleleft y^M \cong y^{M\times M}$ can be identified with the binary operation $*\colon M \times M \to M$. (We don't have to worry about codomains, since there's only one possible codomain to choose from.) In this way, every monoid $(M, e, *)$ in **Set** gives rise to a representable comonoid (y^M, ϵ, δ) in **Poly**. We can just as easily invert this construction, obtaining a monoid for every representable comonoid by taking the underlying set to be the carrier's set of directions, the unit to be the direction picked out by the erasure, and the binary operation to be the duplicator's on-directions function.

Exercise 7.41 (Solution here). Verify Example 7.40 by showing that $(M, e, *)$ satisfies the unitality and associativity requirements of a monoid in $(\mathbf{Set}, 1, \times)$ if and only if (y^M, ϵ, δ) satisfies the erasure and coassociativity requirements of a comonoid in $(\mathbf{Poly}, y, \triangleleft)$. $\Diamond$

Example 7.42 (Cyclic lists). For any $n \in \mathbb{N}$, consider $\mathbb{Z}/n\mathbb{Z}$, the cyclic group of order n, viewed as a monoid or, equivalently, a 1-object category. Its carrier is $y^{\mathbb{Z}/n\mathbb{Z}}$.

As a polynomial functor, $y^{\mathbb{Z}/n\mathbb{Z}}$ sends each set X to the set of length-n tuples in X. But the comonoid structure lets us think of these tuples as *cyclic lists*: once we reach the last element, we can loop back around to the first element. Indeed, as a natural transformation, $\epsilon\colon y^{\mathbb{Z}/n\mathbb{Z}} \to y$ picks out the "current" element via its X-component $\epsilon \triangleleft X\colon y^{\mathbb{Z}/n\mathbb{Z}} \triangleleft X \to y \triangleleft X$, which is just a function $\epsilon_X\colon X^{\mathbb{Z}/n\mathbb{Z}} \to X$; and δ lets us move around the list.

We will see later that comonoids are closed under coproducts, so $\sum_{n\in\mathbb{N}} y^{\mathbb{Z}/n\mathbb{Z}}$ is also a comonoid.

Example 7.43 (Monoid actions). Suppose that $(M, e, *)$ is a monoid, S is a set, and $\alpha\colon S \times M \to S$ is a *(right) monoid action*. That is, for all $s \in S$ we have $\alpha(s, e) = s$ and $\alpha(s, m * n) = \alpha(\alpha(s, m), n)$ for $m, n \in M$; equivalently, the diagrams

$$
\begin{tikzcd}
S \times 1 \arrow[r, "S \times e"] \arrow[dr, equal] & S \times M \arrow[d, "\alpha"] \\
& S
\end{tikzcd}
\qquad \text{and} \qquad
\begin{tikzcd}
S \times M \times M \arrow[r, "S \times *"] \arrow[d, "\alpha \times M"'] & S \times M \arrow[d, "\alpha"] \\
S \times M \arrow[r, "\alpha"] & S
\end{tikzcd}
$$

commute.

Then there is an associated category $\mathcal{MA}$ with objects in S and morphisms $s \xrightarrow{m} \alpha(s, m)$ for each $s \in S$ and $m \in M$. This, in turn, corresponds to a comonoid (Sy^M, ϵ, δ), as we will see in the next exercise.

Exercise 7.44 (Solution here). With notation as in Example 7.43, characterize the comonoid structure on Sy^M.

1. How can we define the erasure ϵ?
2. How can we define the duplicator δ?
3. Verify that the erasure laws hold.
4. Verify that the coassociative law holds.
5. Describe the corresponding category $\mathcal{MA}$. In particular, what are the morphisms between any fixed pair of objects, what are the identity morphisms, and how do morphisms compose?
6. 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 7.38? ◊

Example 7.45 (The category of B-streams). Fix a set B. The set $B^{\mathbb{N}}$ consists of countable sequences of elements in B, which we will call *B-streams*. We can write an B-stream $\overline{b} \in B^{\mathbb{N}}$ as

$$\overline{b} \coloneqq (b_0 \to b_1 \to b_2 \to b_3 \to \cdots),$$

with $b_n \in B$ for each $n \in \mathbb{N}$.

Then there is a monoid action $\tau\colon B^{\mathbb{N}} \times \mathbb{N} \to B^{\mathbb{N}}$ for which

$$\tau(\overline{b}, n) \coloneqq (b_n \to b_{n+1} \to b_{n+2} \to b_{n+3} \to \cdots).$$

Roughly speaking, $\mathbb{N}$ acts on B-streams by shifting them forward by a natural number of steps. We can check that this is a monoid action by observing that $\tau(\overline{b}, 0) = \overline{b}$ and that $\tau(\overline{b}, m + n) = \tau(\tau(\overline{b}, m), n)$.

So by Example 7.43, the corresponding comonoid is carried by $B^{\mathbb{N}}y^{\mathbb{N}}$. Each B-stream $\overline{b}$ is a position, and each $n \in \mathbb{N}$ is a direction at $\overline{b}$ that can be visualized as the sequence of n arrows starting from b_0 and ending at b_n. Then at the end of the direction n is a new B-stream: the rest of $\overline{b}$ starting at b_n. Indeed, this B-stream is exactly $\tau(\overline{b}, n)$, the codomain assigned to the direction n at $\overline{b}$.

Alternatively, if we shift from the domain-centric perspective to the usual hom-set perspective, this comonoid corresponds to a category whose objects are B-streams and whose morphisms $\overline{b} \to \overline{b'}$ consist of every way in which $\overline{b'}$ can be viewed as a continguous substream of $\overline{b}$: that is, there is a morphism $n\colon \overline{b} \to \overline{b'}$ for each $n \in \mathbb{N}$ satisfying

$$(b_n \to b_{n+1} \to \cdots) = (b'_0 \to b'_1 \to \cdots).$$

The identity on $\overline{b}$ is given by $0\colon \overline{b} \to \overline{b}$; and the composite of two morphisms is the sum of the corresponding natural numbers, as a substream of a substream of $\overline{b}$ is just a substream of $\overline{b}$ shifted by the appropriate amount.

We will see this category again in Example 8.38.

Exercise 7.46 (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 concretely, it is 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 the corresponding category a groupoid? ◊

The degree of an object

We could continue to list examples of polynomial comonoids, but of course any list of small categories is already a list of such comonoids. So instead, we conclude this section with some terminology that the polynomial perspective on a category affords.

Definition 7.47 (Degree, linear)**.** Let $\mathcal{C}$ be a category and $c \in \operatorname{Ob}\mathcal{C}$ 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*. If $\deg(c) \cong \mathsf{n}$ for $n \in \mathbb{N}$, we say c has *degree* n.

Exercise 7.48 (Solution here).

1. If every object in $\mathcal{C}$ is linear, what can we say 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. Up to isomorphism, 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? ◊

7.3 Morphisms of polynomial comonoids are retrofunctors

Now that we have characterized the comonoids of **Poly**, let us consider the morphisms between them. These turn out to correspond to a rather odd kind of map between categories known as a *retrofunctor*. [6]

7.3.1 Introducing comonoid morphisms and retrofunctors

First, let us define morphisms of comonoids in the most general setting. If you've seen the definition of a monoid homomorphism (or even a group homomorphism), then this definition may look familiar.

[6] Many authors have referred to these as *cofunctors*, including ourselves in other work and in early versions of this book. However, the prefix *co* in category theory is very special—having to do with taking opposites—and we will see in Remark 7.59 that comonoid homomorphisms are not just opposite-functors.. Thus to keep the prefix *co* more pristine, and in solidarity with other researchers, we have decided to use the term *retrofunctor*, which is an appropriate usage of the term defined by Bob Paré [Par23].

Definition 7.49 (Comonoid morphism)**.** Given a monoidal category $(\mathcal{C}, y, \triangleleft)$ with comonoids $\mathscr{C} := (\mathfrak{c}, \epsilon, \delta)$ and $\mathscr{C}' := (\mathfrak{c}', \epsilon', \delta')$, a *comonoid morphism* (or *morphism of comonoids*) $\mathscr{C} \to \mathscr{C}'$ is a morphism $F\colon \mathfrak{c} \to \mathfrak{c}'$ in $\mathcal{C}$ for which the following diagrams commute:

$$\begin{tikzcd}
\mathfrak{c} \arrow[r, "F"] \arrow[d, "\epsilon"'] & \mathfrak{c}' \arrow[d, "\epsilon'"] \\
y \arrow[r, equal] & y,
\end{tikzcd} \tag{7.50}$$

called the *eraser preservation law* (we say F *preserves erasure*); and

$$\begin{tikzcd}
\mathfrak{c} \arrow[r, "F"] \arrow[d, "\delta"'] & \mathfrak{c}' \arrow[d, "\delta'"] \\
\mathfrak{c} \triangleleft \mathfrak{c} \arrow[r, "F \triangleleft F"'] & \mathfrak{c}' \triangleleft \mathfrak{c}'
\end{tikzcd} \tag{7.51}$$

called the *duplicator preservation law* (we say F *preserves duplication*). We may also say that F *preserves the comonoid structure*.

When the monoidal structure on $\mathcal{C}$ can be inferred, we let **Comon**$(\mathcal{C})$ denote the subcategory of $\mathcal{C}$ whose objects are comonoids in $\mathcal{C}$ and whose morphisms are comonoid morphisms.

So when our monoidal category of interest is $(\mathbf{Poly}, y, \triangleleft)$, a morphism between polynomial comonoids is just a special kind of lens between their carriers that preserves erasure and duplication.

Exercise 7.52 (Solution here). There is something to be proved in the definition above: that comonoids and comonoid morphisms really do form a category. Using the notation from Definition 7.49, verify the following:

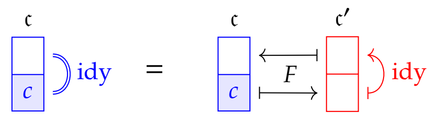
1. The identity morphism on a comonoid is a comonoid morphism.
2. The composite of two comonoid morphisms is a comonoid morphism.

This will show that **Comon**$(\mathcal{C})$ is indeed a subcategory of $\mathcal{C}$. $\Diamond$

Notice that we were very careful in how we stated Theorem 7.28: while we asserted the existence of an isomorphism-preserving one-to-one correspondence between the objects of **Comon**(**Poly**) and **Cat**, we never claimed that these two categories are isomorphic or even equivalent. The strange truth of the matter is that they are not: polynomial comonoid morphisms correspond not to functors, but to different maps of categories called *retrofunctors*.

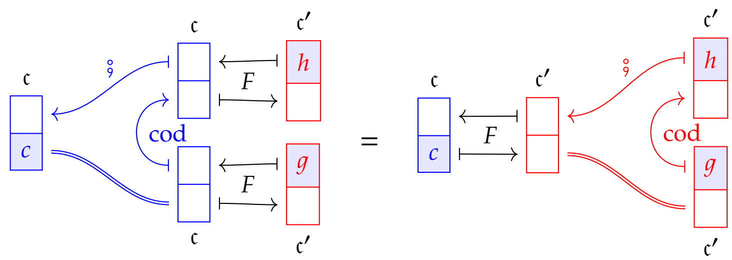
How exactly do these maps behave? If we specify Definition 7.49 to the case of

$(\mathbf{Poly}, y, \triangleleft)$, we can write the eraser preservation law (7.50) in polyboxes as



(7.53)

and the duplicator preservation law (7.51) in polyboxes as



(7.54)

If we read off the equations from these polyboxes, interpreting polynomial comonoids as categories, we derive the following definition of a retrofunctor. (Here (7.56) is equivalent to (7.53), while (7.57) and (7.58) are together equivalent to (7.54).)

Definition 7.55 (Retrofunctor)**.** Let $\mathcal{C}$ and $\mathcal{C}'$ be (small) categories. A *retrofunctor* $F\colon \mathcal{C} \nrightarrow \mathcal{C}'$ consists of

- a function $F\colon \operatorname{Ob}\mathcal{C} \to \operatorname{Ob}\mathcal{C}'$ *forward on objects*[7] and
- a function $F_c^\sharp\colon \mathcal{C}'[Fc] \to \mathcal{C}[c]$ *backward on morphisms* for each $c \in \operatorname{Ob}\mathcal{C}$,

satisfying the following conditions, collectively known as the *retrofunctor laws*:

i. *F preserves identities*:
$$F_c^\sharp \operatorname{id}_{Fc} = \operatorname{id}_c \tag{7.56}$$
for each $c \in \operatorname{Ob}\mathcal{C}$;

ii. *F preserves codomains*:
$$F \operatorname{cod} F_c^\sharp g = \operatorname{cod} g \tag{7.57}$$
for each $c \in \operatorname{Ob}\mathcal{C}$ and $g \in \mathcal{C}'[Fc]$;

iii. *F preserves composites*:[8]
$$F_c^\sharp g \mathbin{⨟} F^\sharp_{\operatorname{cod} F_c^\sharp g} h = F_c^\sharp \left(g \mathbin{⨟} h\right) \tag{7.58}$$
for each $c \in \operatorname{Ob}\mathcal{C}$, $g \in \mathcal{C}'[Fc]$, and $h \in \mathcal{C}'[\operatorname{cod} g]$.

We let $\mathbf{Cat}^\sharp \cong \mathbf{Comon}(\mathbf{Poly})$ denote the category of (small) categories and retrofunctors.

[7]In keeping with standard functor notation, we omit the usual subscript 1 that we include for on-positions (in this case, on-objects) functions. We often omit parentheses when applying these functions as well.

[8]In particular, the codomains of either side of (7.58) are equal. This isn't actually guaranteed by the other laws, so it is worth noting on its own; see for example the proof of Proposition 7.61.

Remark 7.59. For experts, we explain the term *retrofunctor* from Definition 7.55. Let $\mathcal{C}, \mathcal{C}'$ be categories, and consider them as monads in $\mathbb{S}\mathbf{pan}$. A functor between them consists of a function $F_{\mathrm{Ob}} \colon \mathrm{Ob}(\mathcal{C}) \to \mathrm{Ob}(\mathcal{C}')$ and a 2-cell, as shown left

$$
\begin{tikzcd}
\mathrm{Ob}(\mathcal{C}) \arrow[r, "\mathrm{Mor}(\mathcal{C})", "|"] \arrow[d, "F_{\mathrm{Ob}}"'] \arrow[dr, phantom, "\Downarrow F_{\mathrm{Mor}}"] & \mathrm{Ob}(\mathcal{C}) \arrow[d, "F_{\mathrm{Ob}}"] \\
\mathrm{Ob}(\mathcal{C}') \arrow[r, "\mathrm{Mor}(\mathcal{C}')"', "|"] & \mathrm{Ob}(\mathcal{C}')
\end{tikzcd}
\qquad
\begin{tikzcd}[column sep=huge]
\mathrm{Ob}(\mathcal{C}) \arrow[r, bend left, "\mathrm{Mor}(\mathcal{C})", "|"{name=U}] \arrow[r, bend right, "\check{F}_{\mathrm{Ob}} \fatsemi \mathrm{Mor}(\mathcal{C}') \fatsemi \widehat{F_{\mathrm{Ob}}}"', "|"{name=D}] \arrow[from=U, to=D, phantom, "\Downarrow F_{\mathrm{Mor}}"] & \mathrm{Ob}(\mathcal{C})
\end{tikzcd}
$$

satisfying the properties of a monad homomorphism.

In [Par23, Definition 6.1], Paré defines *retromorphism of monads* in double categories like $\mathbb{S}\mathbf{pan}$. We will not discuss the more general definition here, but in a framed bicategory (equipment), where we have companions $\check{f}$ and conjoints $\widehat{f}$ of tight maps f, it is easy to check that monads lift along tight morphisms in the sense that the horizontal cell $\left(\check{F}_{\mathrm{Ob}} \fatsemi \mathrm{Mor}(\mathcal{C}') \fatsemi \widehat{F_{\mathrm{Ob}}}\right) \colon \mathrm{Ob}(\mathcal{C}) \twoheadrightarrow \mathrm{Ob}(\mathcal{C})$ is a monad if $\mathrm{Mor}(\mathcal{C}')$ is. Hence, a functor is equivalently given by a function $F_{\mathrm{Ob}} \colon \mathrm{Ob}(\mathcal{C}) \to \mathrm{Ob}(\mathcal{C}')$ such that the 2-cell shown above right is a monad homomorphism.

A retromorphism of monads—in our case, a *retrofunctor*—is simply a monad map going the other way:[9]

$$
\begin{tikzcd}[column sep=huge]
\mathrm{Ob}(\mathcal{C}) \arrow[r, bend left, "\mathrm{Mor}(\mathcal{C})", "|"{name=U}] \arrow[r, bend right, "\check{F}_{\mathrm{Ob}} \fatsemi \mathrm{Mor}(\mathcal{C}') \fatsemi \widehat{F_{\mathrm{Ob}}}"', "|"{name=D}] \arrow[from=U, to=D, phantom, "\Uparrow"] & \mathrm{Ob}(\mathcal{C})
\end{tikzcd}
$$

Henceforth we will identify the category $\mathbf{Cat}^\sharp$ with the isomorphic category $\mathbf{Comon}(\mathbf{Poly})$, eliding the difference between comonoids in $\mathbf{Poly}$ and categories.

Since each retrofunctor includes a lens between its carrier polynomials, retrofunctors compose the way lenses do.

Exercise 7.60 (Solution here). Let $\mathcal{C}, \mathcal{D}, \mathcal{E}$ be categories, and let $F \colon \mathcal{C} \nrightarrow \mathcal{D}$ and $G \colon \mathcal{D} \nrightarrow \mathcal{E}$ be retrofunctors between them.

1. Characterize the behavior of the identity retrofunctor $\mathrm{id}_{\mathcal{D}}$ on $\mathcal{D}$. Where does it send each object? Where does it send each morphism?
2. Characterize the behavior of the composite retrofunctor $F \fatsemi G$. Where does it send each object and morphism? ◊

On the surface, functors and retrofunctors have much in common: both send objects to objects and morphisms to morphisms in a way that preserves domains and codomains as well as identities and composites. The main difference is that functors

[9] As a hint to the connection in terms of the (yet-undefined) double category $\mathbb{C}\mathbf{at}^\sharp$, note that when the monads in question are left adjoints, their right adjoints will automatically be comonads, and a morphism between these comonads will be a retromorphism between the monads.

send morphisms *forward*, while retrofunctors send morphisms *backward*. As we work with retrofunctors, it will be helpful to remember the following:

A retrofunctor F goes forward on objects and backward on morphisms.
Codomains are objects, so F preserves them going forward.
Identities and composites are morphisms, so F preserves them going backward.

Before we explore just how different functors and retrofunctors can be, let us note a few more similarities that these two kinds of maps between categories share. For example, retrofunctors, like functors, preserve isomorphisms.

Proposition 7.61. Let $F\colon \mathcal{C} \nrightarrow \mathcal{D}$ be a retrofunctor, $c \in \mathcal{C}$ be an object, and $g\colon Fc \to \_$ be an isomorphism in $\mathcal{D}$. Then $F^\sharp_c g$ is also an isomorphism in $\mathcal{C}$.

Proof. Let $c' := \operatorname{cod} F^\sharp_c g$, so that $Fc' = \operatorname{cod} g$ by (7.57), and let $g^{-1}\colon Fc' \to Fc$ be the inverse of g. Then

$$
\begin{aligned}
\mathrm{id}_c &= F^\sharp_c\, \mathrm{id}_{Fc} && (7.56)\\
&= F^\sharp_c\left(g \mathbin{⨾} g^{-1}\right)\\
&= F^\sharp_c g \mathbin{⨾} F^\sharp_{c'}\left(g^{-1}\right), && (7.58)
\end{aligned}
$$

so in particular $c = \operatorname{cod}\mathrm{id}_c = \operatorname{cod} F^\sharp_{c'}\left(g^{-1}\right)$, and

$$
\begin{aligned}
\mathrm{id}_{c'} &= F^\sharp_{c'}\, \mathrm{id}_{Fc'} && (7.56)\\
&= F^\sharp_{c'}\left(g^{-1} \mathbin{⨾} g\right)\\
&= F^\sharp_{c'}\left(g^{-1}\right) \mathbin{⨾} F^\sharp_{\operatorname{cod} F^\sharp_{c'}\left(g^{-1}\right)} g && (7.58)\\
&= F^\sharp_{c'}\left(g^{-1}\right) \mathbin{⨾} F^\sharp_c g.
\end{aligned}
$$

Hence $F^\sharp_c g$ and $F^\sharp_{c'}\left(g^{-1}\right)$ are inverses, and the result follows. □

Moreover, isomorphisms in **Cat** correspond to isomorphisms in $\mathbf{Cat}^\sharp$.

Exercise 7.62 (Solution here). We've justified the "isomorphism-preserving" part of Theorem 7.28 implicitly, but let's make it explicit.

Recall that two categories $\mathcal{C}$ and $\mathcal{D}$ are isomorphic in **Cat** if there exist functors $F\colon \mathcal{C} \to \mathcal{D}$ and $G\colon \mathcal{D} \to \mathcal{C}$ that are mutually inverse, i.e. $F \mathbin{⨾} G$ and $G \mathbin{⨾} F$ are identity functors on $\mathcal{C}$ and $\mathcal{D}$. Similarly, $\mathcal{C}$ and $\mathcal{D}$ are isomorphic in $\mathbf{Cat}^\sharp$ if there exist retrofunctors $H\colon \mathcal{C} \nrightarrow \mathcal{D}$ and $K\colon \mathcal{D} \nrightarrow \mathcal{C}$ that are mutually inverse, i.e. $H \mathbin{⨾} K$ and $K \mathbin{⨾} H$ are identity retrofunctors on $\mathcal{C}$ and $\mathcal{D}$. Show that $\mathcal{C}$ and $\mathcal{D}$ are isomorphic in **Cat** if and only if they are isomorphic in $\mathbf{Cat}^\sharp$. ◊

But while isomorphisms in $\mathbf{Cat}^\sharp$ are the same as those in **Cat**, the non-isomorphisms can be very different. We'll see this in the examples to come.

7.3.2 Examples of retrofunctors

From a realm where functors reign supreme, the back-and-forth behavior of retrofunctors can seem foreign and counterintuitive. Whereas a functor $\mathcal{C} \to \mathcal{D}$ can be thought of as a *diagram*—a picture in the shape of $\mathcal{C}$, drawn with the objects and arrows of $\mathcal{D}$—retrofunctors are much more like the dynamical systems of Chapter 4.[10]

That is, a retrofunctor $F\colon \mathcal{C} \nrightarrow \mathcal{D}$ is a way of interacting with the states (objects) and transitions (morphisms) within $\mathcal{C}$ by way of $\mathcal{D}$. Imagine the retrofunctor as a box, with $\mathcal{C}$ on the inside and $\mathcal{D}$ on the outside. Some $c \in \mathcal{C}$ may be the current state inside the box, but all anyone outside the box can see is the object $Fc \in \mathcal{D}$ that the box chooses to display in lieu of c. Still, any transition g out of Fc can be selected from the outside; the box guarantees that whatever c is on the inside, there is a corresponding transition $F_c^\sharp g$ out of that c. As g is followed from Fc to $\operatorname{cod} g$ on the outside, $F_c^\sharp g$ is followed from c to $\operatorname{cod} F_c^\sharp g$ on the inside. But codomain preservation guarantees that the new state $\operatorname{cod} g$ on the outside is equal to what the box would want to display in lieu of the new state $\operatorname{cod} F_c^\sharp g$ on the inside, as $\operatorname{cod} g = F \operatorname{cod} F_c^\sharp g$, Then the process repeats in a manner compatible with identities and composition.

Here we give a variety of examples of retrofunctors to get a better handle on them. Often we will denote a category by its carrier when its comonoid structure can be inferred from context, and $\mathcal{C}$ will be a category throughout with carrier $\mathfrak{c}$.

Retrofunctors to preorders

Given a retrofunctor from $\mathcal{C}$ to a preorder $\mathcal{P}$, we can think of $\mathcal{P}$ as providing a simplified model or abstraction of the states and transitions possible in $\mathcal{C}$, picking canonical transitions in $\mathcal{C}$ along the way to exhibit the model. While the transitions in a general category may be more complex, all that a preorder tells you is whether you can get from one state to another or not. Let's see some examples.

Example 7.63 (Retrofunctors to discrete categories). The discrete category on a set S is the category with objects in S and only identity morphisms; its carrier is $S\mathcal{y}$. So a retrofunctor $F\colon \mathcal{C} \nrightarrow S\mathcal{y}$ is completely determined by its behavior on objects: to preserve identities, it can only send the morphisms in $S\mathcal{y}$ back to the identity morphisms in $\mathcal{C}$. We can identify F with a function $\operatorname{Ob} \mathcal{C} \to S$, assigning each state in $\mathcal{C}$ a label in S without revealing anything about the transitions between them.

Exercise 7.64 (Solution here).

1. Show that $\mathcal{y}$ has a unique comonoid structure.
2. Show that $\mathcal{y}$ with its comonoid structure is terminal in $\mathbf{Cat}^\sharp$.
3. Explain why $\mathcal{y}$ is terminal using the language of states and transitions. ◊

[10] In fact, we will see in Chapter 8 that retrofunctors generalize our dynamical systems.

Example 7.65 (Retrofunctors to the walking arrow). Consider a retrofunctor $F\colon \mathcal{C} \nrightarrow \mathcal{A}$, where $\mathcal{A}$ is the walking arrow category

$$\mathcal{A} \coloneqq \boxed{s \xrightarrow{a} t}$$

from Example 7.34. On objects, F is a function $\mathrm{Ob}\,\mathcal{C} \to \{s, t\}$, so each object of $\mathcal{C}$ lies in either $\mathcal{C}_s \coloneqq F^{-1}s$ or $\mathcal{C}_t \coloneqq F^{-1}t$ (but not both). Then on morphisms, preservation of identities determines where $F^\sharp$ sends id_s and id_t, while preservation of codomains ensures that for each $c \in \mathcal{C}_s$, the morphism $F^\sharp_c a\colon a \to \_$ that $F^\sharp$ sends a back to must satisfy

$$F\,\mathrm{cod}\,F^\sharp_c a = \mathrm{cod}\,a = t$$

and thus $\mathrm{cod}\,F^\sharp_c a \in \mathcal{C}_t$. In particular, for every object $c \in \mathcal{C}$ that F sends to s, there must be at least one morphism from c to an object that F sends to t, so that one of those morphisms can be $F^\sharp_c a$. As there are no nontrivial composites in $\mathcal{A}$, the retrofunctor F automatically preserves composites.

In summary, a retrofunctor $F\colon \mathcal{C} \nrightarrow \mathcal{A}$ divides the objects of $\mathcal{C}$ between $\mathcal{C}_s$ and $\mathcal{C}_t$ and fixes a morphism from each object in $\mathcal{C}_s$ to some object in $\mathcal{C}_t$. We can think of F as separating the states of $\mathcal{C}$ into source states and target states, modeled by the s state and the t state in $\mathcal{A}$, respectively; then every source state is assigned a target state and a way of getting to that target state via a transition in $\mathcal{C}$.

Given a retrofunctor $F\colon \mathcal{C} \nrightarrow \mathcal{D}$ and an object $d \in \mathcal{D}$, we will continue to use the notation $\mathcal{C}_d \coloneqq F^{-1}d$ to denote the set of objects in $\mathcal{C}$ that F sends to d.

Exercise 7.66 (Solution here). Let $F\colon \mathcal{C} \nrightarrow \mathcal{A}$ be a retrofunctor from $\mathcal{C}$ to the walking arrow category $\mathcal{A}$, as in Example 7.65. If $Fc = s$ for all $c \in \mathcal{C}$, what can we say about $\mathcal{C}$? ◊

Exercise 7.67 (Solution here).

1. Recall the star-shaped category $\mathsf{y}^{\mathsf{n}+1} + \mathsf{n}\mathcal{y}$ from Exercise 7.37. Describe retrofunctors to it.
2. Describe retrofunctors to the preorder $(\mathbb{N}, \leq)$, viewed as a category: its objects are natural numbers, and there is a morphism $m \to n$ if and only if $m \leq n$.
3. Describe retrofunctors to the preorder $(\mathbb{N}, \geq)$: its objects are natural numbers, and there is a morphism $n \to m$ if and only if $n \geq m$. ◊

Example 7.68 (Retrofunctors to the walking commutative square). Consider a retrofunctor $F\colon \mathcal{C} \nrightarrow \mathcal{CS}$, where $\mathcal{CS}$ is the *walking commutative square category*

$$\mathcal{CS} := \boxed{\begin{array}{c}\begin{tikzcd} w \arrow[r, "h"] \arrow[d, "f"'] & y \arrow[d, "k"] \\ x \arrow[r, "g"'] & z \end{tikzcd} \\ f \mathbin{⨾} g = h \mathbin{⨾} k\end{array}}$$

On objects, F is a function $\operatorname{Ob}\mathcal{C} \to \{w, x, y, z\}$, so each object of $\mathcal{C}$ lies in exactly one of $\mathcal{C}_w, \mathcal{C}_x, \mathcal{C}_y$, and $\mathcal{C}_z$. Then on morphisms, out of every object $X \in \mathcal{C}_x$ there is a morphism $F^\sharp_X g \colon X \to \_$ to an object in $\mathcal{C}_z$, and out of every object $Y \in \mathcal{C}_y$ there is a morphism $F^\sharp_Y k \colon Y \to \_$ also to an object in $\mathcal{C}_z$. Finally, out of every object $W \in \mathcal{C}_w$ there is a morphism $F^\sharp_W f \colon W \to X_W$ to an object $X_W \in \mathcal{C}_x$ and a morphism $F^\sharp_W h \colon W \to Y_W$ to an object in $Y_W \in \mathcal{C}_y$. As F preserves composites, these must all then satisfy

$$F^\sharp_W f \mathbin{⨾} F^\sharp_{X_W} g = F^\sharp(f \mathbin{⨾} g) = F^\sharp(h \mathbin{⨾} k) = F^\sharp_W h \mathbin{⨾} F^\sharp_{Y_W} k;$$

in particular, $F^\sharp_{X_W} g$ and $F^\sharp_{Y_W} k$ must share a common codomain $Z_W \in \mathcal{C}_z$, yielding the following commutative square in $\mathcal{C}$:

$$\begin{tikzcd} W \arrow[r, "F^\sharp_W h"] \arrow[d, "F^\sharp_W f"'] & Y_W \arrow[d, "F^\sharp_{Y_W} k"] \\ X_W \arrow[r, "F^\sharp_{X_W} g"'] & Z_W. \end{tikzcd}$$

Exercise 7.69 (Solution here). Let $\mathcal{A}$ denote the walking arrow category, as in Example 7.65, and let $\mathcal{CS}$ denote the walking commutative square category, as in Example 7.68.

1. List the retrofunctors $\mathcal{CS} \nrightarrow \mathcal{A}$.
2. List the retrofunctors $\mathcal{A} \nrightarrow \mathcal{CS}$. ◊

Exercise 7.70 (Solution here).

1. What does a retrofunctor from y to a poset represent?
2. Consider the chain poset $[n] \cong \sum_{i=0}^{n} y^{i+1}$. How many retrofunctors are there from $[m] \to [n]$ for all $m \leq n$? ◊

Retrofunctors to monoids

When a monoid $(M, e, *)$, viewed as a 1-object category y^M, is the codomain of a retrofunctor $\mathcal{C} \nrightarrow y^M$, it plays the role of a joystick: an "input device" that "reports its. . . direction to the device it is controlling."[11] Like a joystick, y^M stays in one "place"—a single state—but has a number of directions it can take that are reported back to $\mathcal{C}$, controlling the way it moves through its transitions. As we string together a sequence of directions in M, we chart a course through the transitions of $\mathcal{C}$. We make this analogy concrete in the following examples.

Example 7.71 (Arrow fields). Consider the monoid $(\mathbb{N}, 0, +)$ viewed as a category $y^{\mathbb{N}}$. Retrofunctors $\mathcal{C} \nrightarrow y^{\mathbb{N}}$ have been called *admissible sections* [Agu97]. We prefer to call them *arrow fields* (on $\mathcal{C}$), for they turn out to resemble vector fields—but with arrows in $\mathcal{C}$ instead of vectors.[12] We'll have more to say about these in Theorem 8.59, but our goal here is simply to unpack the definition.

To specify a retrofunctor $A\colon \mathcal{C} \nrightarrow y^{\mathbb{N}}$, we first say what it does on objects, but this is already decided: there is only one object in $y^{\mathbb{N}}$, so every object of $\mathcal{C}$ is sent to it. This also means that codomains are automatically preserved. So as will be the case for all retrofunctors to monoids, A is characterized by its behavior on morphisms: for each object $c \in \mathcal{C}$, the retrofunctor assigns each $n \in \mathbb{N}$ a morphism $A^\sharp_c n$ of $\mathcal{C}$ emanating from c. That's a lot of data, but we still have two retrofunctor laws to pare it down:

$$A^\sharp_c 0 = \mathrm{id}_c \qquad \text{and} \qquad A^\sharp_c(m+n) = A^\sharp_c m \mathbin{⨾} A^\sharp_{\operatorname{cod} A^\sharp_c m} n.$$

Then for each $c \in \mathcal{C}$, since every $n \in \mathbb{N}$ is a sum of 1's, the morphism $A^\sharp_i n$ can be decomposed into n copies of $A^\sharp_{c_j} 1$ for objects $c_0 \coloneqq c, c_1, \ldots, c_n \in \mathcal{C}$, as follows:

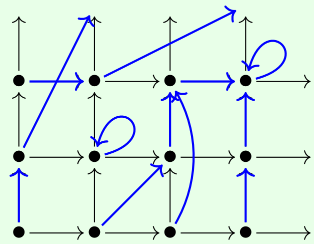
$$c = c_0 \xrightarrow{A^\sharp_{c_0} 1} c_1 \xrightarrow{A^\sharp_{c_1} 1} \cdots \xrightarrow{A^\sharp_{c_{n-1}} 1} c_n. \tag{7.72}$$

Here each $c_{j+1} \coloneqq \operatorname{cod} A^\sharp_{c_j} 1$.

Thus an arrow field of $\mathcal{C}$ is given by independently choosing a morphism emanting from each $c \in \mathcal{C}$ to be $A^\sharp_c 1$: an arrow (morphism) out of each object, like how a vector field has a vector out of each point. Indeed, any such choice uniquely determines the retrofunctor $A\colon \mathcal{C} \nrightarrow y^{\mathbb{N}}$: as every object is assigned an arrow coming out of it, we can follow the arrow out of c to an object c_1, then following the arrow out of c_1 to an object c_2, and so on until we have followed the n arrows in (7.72), which then compose to yield $A^\sharp_c n$. With our joystick analogy, a single flick sends us from a state $c \in \mathcal{C}$ along its assigned arrow $A^\sharp_c 1$, while n flicks send us through n arrows along the arrow field.

[11] Description from Wikipedia.

Here is an example of an arrow field on the product of preorders $(\mathbb{N}, \leq) \times (\mathbb{N}, \leq)$:



Every object has been assigned an emanating morphism drawn in blue, but there need not be any rhyme or reason to our choice.

Exercise 7.73 (Solution here). How many arrow fields on the category $\boxed{\bullet \to \bullet}$ are there? $\lozenge$

We will see later in Proposition 8.60 that the arrow fields 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 7.74 (Solution here). Consider the monoid of integers $(\mathbb{Z}, 0, +)$ as a 1-object category $y^{\mathbb{Z}}$, and let $y^{\mathbb{N}}$ be the monoid of natural numbers $(\mathbb{N}, 0, +)$ viewed as a 1-object category as above.

1. Describe the data of a retrofunctor $\mathcal{C} \nrightarrow y^{\mathbb{Z}}$.
2. What would you say is the canonical retrofunctor $y^{\mathbb{Z}} \nrightarrow y^{\mathbb{N}}$? $\lozenge$

Exercise 7.75 (Solution here).

1. Suppose that M, N are monoids (each is a category with one object). Are retrofunctors 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 retrofunctor. Does there necessarily exist a retrofunctor $\mathcal{C}^{\mathrm{op}} \nrightarrow \mathcal{D}^{\mathrm{op}}$ that acts the same as F on objects? $\lozenge$

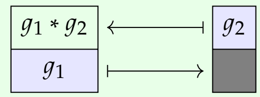
Exercise 7.76 (Monoid actions; solution here). Recall from Example 7.43 that every monoid action $\alpha \colon S \times M \to S$, where S is a set and $(M, e, *)$ is a monoid, gives rise to a category carried by Sy^M. Show that the projection $Sy^M \to y^M$ is a retrofunctor. $\lozenge$

Example 7.77. Let $(G, e, *)$ be a group and (y^G, ϵ, δ) the corresponding comonoid. There

---

[12] After all, a vector field is a section of a vector bundle. Our arrow fields will be sections of $\mathcal{C}$'s carrier, viewed as a bundle of directions over positions.

is a retrofunctor $Gy^G \nrightarrow y^G$ given by



To see this is a retrofunctor, 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 7.78 (Solution here). Does the idea of Example 7.77 work when G is merely a monoid, or does something go subtly wrong somehow? ◊

Proposition 7.79. There is a fully faithful functor $\mathbf{Mon}^{\mathrm{op}} \to \mathbf{Cat}^\sharp$, whose image consists of all categories whose carriers are representable.

Proof. Given a monoid $(M, e, *)$, we think of it as a category with one object; its carrier y^M is representable. A retrofunctor between such categories carries no data in its on-objects part, and codomains are automatically preserved. Retrofunctors $y^M \to y^N$ simply carry elements of N to elements of M, preserving identity and composition, exactly the description of monoid homomorphisms. □

Proposition 7.80. There is an adjunction

$$\mathbf{Cat}^\sharp(\mathcal{C}, Ay) \cong \mathbf{Set}(\mathrm{Ob}\,\mathcal{C}, A)$$

for $\mathcal{C} \in \mathbf{Cat}^\sharp$ and $A \in \mathbf{Set}$.

Proof. In the solution to Exercise 7.36, we saw that a category is discrete iff its carrier is a linear polynomial: this occurs when the only arrow emanating from each object is its identity. Thus Ay corresponds to a discrete category. A retrofunctor from any category to a discrete category needs to say what happens on objects, but the rest of the data is determined because identities need to be sent back to identities. This is the content of the proposition. □

Exercise 7.81 (Continuous arrow fields; solution here). Suppose we say that a *continuous arrow field* on $\mathcal{C}$ is a retrofunctor $\mathcal{C} \nrightarrow y^{\mathbb{R}}$, viewing $y^{\mathbb{R}}$ as the monoid of real numbers with addition.

Describe continuous arrow fields in $\mathcal{C}$ using elementary terms, i.e. to someone who doesn't know what a retrofunctor is and isn't yet ready to learn. ◊

Example 7.82 (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 $\mathbb{R}^n$. We are often interested in integrating this vector field to get flow lines, or integral curves. In other words, for each $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, viewed as a point, and each $t \in \mathbb{R}$, viewed as a quantity of time, we can begin at x and move along the vector field for time t, arriving at a new point x^{+t}. These satisfy the equations

$$x^{+0} = x \qquad \text{and} \qquad x^{+t_1+t_2} = (x^{+t_1})^{+t_2}. \tag{7.83}$$

Let's call such things *differentiable dynamical systems* with time domain $(T, 0, +)$; above, we used $T \coloneqq \mathbb{R}$, but any monoid will do.

Dynamical systems in the above sense are retrofunctors $F\colon \mathbb{R}^n y^{\mathbb{R}^n} \nrightarrow y^T$. In order to say this, we first need to say how both $\mathcal{C} \coloneqq \mathbb{R}^n y^{\mathbb{R}^n}$ 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 retrofunctor 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 (7.83) ensures the retrofunctoriality 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

$$F^\sharp(x^{+t_1}, t_2) + F^\sharp(x, t_1) = (x^{+t_1})^{+t_2} - x^{+t_1} + x^{+t_1} - x = x^{+t_1+t_2} - x = F^\sharp(x, t_1 + t_2).$$

Retrofunctors from state categories

By now we should be very familiar with lenses from state categories, which are our original dynamical systems. A retrofunctor from a state category, then, is just a dynamical system that satisfies the retrofunctor laws. It turns out that retrofunctors from state categories are particularly noteworthy: just as a polynomial comonoid $\mathcal{C}$ can be identified with a category, a retrofunctor out of $\mathcal{C}$ can be identified with a number of equivalent categorical constructions on $\mathcal{C}$, perhaps the most familiar being a functor $\mathcal{C} \to \mathbf{Set}$. But these equivalences deserve their own subsection to examine in full; we'll defer them to Section 7.3.3. For now, let's look at some examples of retrofunctors out of state categories.

Example 7.84 (Retrofunctors from state categories to $\mathcal{C}$ are $\mathcal{C}$-coalgebras). Recall from Example 6.67 that for a set S, lenses $Sy^S \to p$ correspond to functions $S \to p(S)$ known as coalgebras for the functor p. As a retrofunctor $Sy^S \nrightarrow \mathcal{C}$ is just a special kind of lens from Sy^S to $\mathfrak{c}$, the carrier of $\mathcal{C}$, it should correspond to a special kind of coalgebra $S \to \mathfrak{c}(S)$ for the functor $\mathfrak{c}$.

Taking $A = B = S \in \mathbf{Set}$ in (6.65), we find that there is a natural isomorphism between dynamical systems $Sy^S \to p$ and functions $S \to p(S)$, also known as a *coalgebra for the functor p* or a *p-coalgebra*.[13]

Example 7.85 (Retrofunctors between state categories are very well-behaved lenses). Our familiar state category on S from Example 7.38 is the category with objects in S and exactly 1 morphism between every pair of objects; when we label each morphism with its codomain, its carrier is Sy^S, the identity of $s \in S$ is s, and (disregarding domains) $s \mathbin{⨟} s' = s'$ for composable $s, s' \in S$.

Then a retrofunctor $Sy^S \nrightarrow Ty^T$ between two state categories corresponds to what is known to functional programmers as a *very well-behaved lens*. We actually defined this way back in Example 3.43, where we called the on-objects (on-positions) function of such a retrofunctor $\text{get}\colon S \to T$, and the on-morphisms (on-directions) function $\text{put}\colon S \times T \to S$.[14] Then the retrofunctor laws are as follows:

1. Preservation of identities (7.56) becomes
$$\text{put}(s, \text{get}(s)) = s,$$
for all $s \in S$, known as the *get-put law* (named in diagrammatic order: we apply get before we apply put).
2. Preservation of codomains (7.57) becomes
$$\text{get}(\text{put}(s, t)) = t,$$
for all $s \in S$ and $t \in T$, known as the *put-get law*.
3. Preservation of composition becomes
$$\text{put}(\text{put}(s, t), t') = \text{put}(s, t')$$

[13] There are two versions of coalgebras we are interested in (and more that we are not) with distinct definitions: a *coalgebra for a functor*, which is the version used here, and a *coalgebra for a comonad*, which is a coalgebra for a functor with extra conditions that we will introduce later.

for all $s \in S$ and $t, t' \in T$, known as the *put-put law*.

In fact, it turns out that these laws can be satisfied 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 retrofunctors $Sy^S \nrightarrow Ty^T$. A stringent condition, no? We'll explore it in Exercise 7.87 below.

Let's explore why retrofunctors between state categories are just product projections. A product projection $A \times B \to A$ always has a second factor B; if every retrofunctor between state categories is a product projection, what is the second factor? It turns out to be

$$U \coloneqq \{u \colon T \to S \mid \forall t, t' \in T,\ \mathrm{get}(u(t)) = t \text{ and } \mathrm{put}(u(t), t') = u(t')\}.$$

In other words, we will show that if (get, put) defines a retrofunctor $Sy^S \nrightarrow Ty^T$, then there is a bijection $S \cong T \times U$ making get: $S \to T$ a product projection. We then prove the converse in Exercise 7.86.

Assume $(\mathrm{get}, \mathrm{put}) \colon Sy^S \nrightarrow Ty^T$ is a retrofunctor, so that it satisfies the enumerated laws. First, we define a function $\alpha \colon S \to T \times U$ as follows. Given $s \in S$, the function $u_s \colon T \to S$ defined by

$$u_s(t) = \mathrm{put}(s, t)$$

lies in U: we check that it satisfies

$$\mathrm{get}(u_s(t)) = \mathrm{get}(\mathrm{put}(s, t)) = t$$

by the put-get law for $t \in T$ and

$$\mathrm{put}(u_s(t), t') = \mathrm{put}(\mathrm{put}(s, t), t') = \mathrm{put}(s, t') = u_s(t')$$

by the put-put law for $t, t' \in T$. We can therefore define a function $\alpha \colon S \to T \times U$ by

$$\alpha(s) = \big(\mathrm{get}(s), u_s\big)\,.$$

In the other direction, we have a function $\beta \colon T \times U \to S$ given by

$$\beta(t, u) = u(t).$$

The two functions α and β are mutually inverse: $\alpha \mathbin{⨟} \beta \colon S \to S$ is the identity because

$$\beta(\alpha(s)) = u_s(\mathrm{get}(s)) = \mathrm{put}(s, \mathrm{get}(s)) = s$$

by the get-put law, while $\beta \mathbin{⨟} \alpha \colon T \times U \to T \times U$ is the identity because

$$\alpha(\beta(t, u)) = \big(\mathrm{get}(u(t)), u_{u(t)}\big) = (t, t' \mapsto \mathrm{put}(u(t), t')) = (t, u),$$

as $\mathrm{get}(u(t)) = t$ and $\mathrm{put}(u(t), t') = u(t')$ by construction for $u \in U$. Thus $S \cong T \times U$, and the product projection $S \xrightarrow{\alpha} T \times U \to T$ sends $s \mapsto \mathrm{get}(s)$, as desired.

We have therefore shown that for every retrofunctor $Sy^S \nrightarrow Ty^T$, there exists a set U for which $S \cong T \times U$ and the on-positions function get is the product projection $S \cong T \times U \to T$. Notice that the on-directions function put can be uniquely recovered from the bijection $S \cong T \times U$ we constructed: it is determined by the functions $u_s \colon T \to S$ for $s \in S$, which in turn is determined by the second projection $T \times U \to U$.

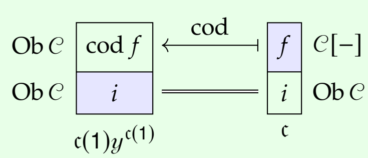
More precisely, composing $\alpha \colon S \to T \times U$ with the projection to U yields a map $S \to U$ sending $s \mapsto u_s$; then put(s, t) is given by $u_s(t)$. Of course, a priori U could just be a set—we may not know how to interpret its elements as a functions $T \to S$. This is where $\beta \colon T \times U \to S$ comes in: we know $\beta(t, u_s) = u_s(t)$. So put(s, t) must be $\beta(t, u_s)$.

Exercise 7.86 (Solution here). Let S, T, U be sets for which we have a bijection $S \cong T \times U$. Show that there exists a unique retrofunctor $Sy^S \nrightarrow Ty^T$ whose on-positions function $S \cong T \times U \to T$ is given by the product projection. ◊

Exercise 7.87 (Solution here).

1. Suppose $|S| = 3$. How many retrofunctors are there $Sy^S \to Sy^S$?
2. Suppose $|S| = 4$ and $|T| = 2$. How many retrofunctors are there $Sy^S \nrightarrow Ty^T$? ◊

Example 7.88. We have a state category $\mathfrak{c}(1)y^{\mathfrak{c}(1)}$ on the set of objects of $\mathcal{C}$. Define a lens $\mathfrak{c}(1)y^{\mathfrak{c}(1)} \to \mathfrak{c}$ by



sending each object $i \in \mathcal{C}$ to itself on positions and, at i, sending each morphism $f \colon i \to \_$ to its codomain cod f on directions.

This lens is a retrofunctor $\mathfrak{c}(1)y^{\mathfrak{c}(1)} \nrightarrow \mathcal{C}$ because it sends identities back to identities, codomains forward to codomains, and preserves composition (trivially, since each morphism in $c(1)y^{c(1)}$ is determined by its domain and codomain.

Exercise 7.89 (Solution here). Fix an object $i \in \mathfrak{c}(1) = \mathrm{Ob}\,\mathcal{C}$. Then we have a state category $\mathfrak{c}[i]y^{\mathfrak{c}[i]}$ on the set $\mathfrak{c}[i] = \mathcal{C}[i]$ of morphisms out of i in $\mathcal{C}$. Define a lens

[14]More precisely, we are treating the on-morphism functions $T \to S$ for each $s \in S$ of a retrofunctor $Sy^S \nrightarrow Ty^T$ as a single function $S \times T \to S$.

$\mathfrak{c}[i]y^{\mathfrak{c}[i]} \to \mathfrak{c}$ by

$$
\begin{array}{r|c|c|c|l}
\cline{2-2}\cline{4-4}
\mathcal{C}[i] & f \mathbin{⨟} g & \xleftarrow{\;⨟\;} & g & \mathcal{C}[-] \\
\cline{2-2}\cline{4-4}
\mathcal{C}[i] & f & \xrightarrow[\text{cod}]{} & \operatorname{cod} f & \operatorname{Ob}\mathcal{C} \\
\cline{2-2}\cline{4-4}
\multicolumn{1}{c}{} & \multicolumn{1}{c}{\mathfrak{c}[i]y^{\mathfrak{c}[i]}} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{\mathfrak{c}} &
\end{array}
$$

sending each morphism $f: i \to \_$ to its codomain on positions and, at f, sending each morphism g: $\operatorname{cod} f \to \_$ to the composite $f \mathbin{⨟} g: i \to \_$ on directions. Is this lens a retrofunctor $\mathfrak{c}[i]y^{\mathfrak{c}[i]} \nrightarrow \mathcal{C}$? ◊

We'll revisit retrofunctors from state categories in Section 7.3.3.

Other retrofunctors

Example 7.90 (Objects aren't representable in $\mathbf{Cat}^\sharp$). In the world of categories and the usual functors between them, the terminal category $\mathcal{T} := \boxed{\bullet}$ with one object and one morphism *represents objects*, in the sense that functors $\mathcal{T} \to \mathcal{C}$ naturally correspond to objects in $\mathcal{C}$.

Unfortunately, the same cannot be said for retrofunctors: we'll see in Exercise 7.91 that there does not exist a fixed category $\mathcal{U}$ for which retrofunctors $\mathcal{U} \nrightarrow \mathcal{C}$ are in bijection with objects in $\mathcal{C}$ for every category $\mathcal{C}$.

Retrofunctors $\mathcal{T} \nrightarrow \mathcal{C}$ are somewhat strange beasts: because they must preserve codomains, 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.

Exercise 7.91 (Solution here). We saw in Exercise 7.27 that $2y$ has a unique comonoid structure.

1. Show that for any category $\mathcal{U}$, retrofunctors $\mathcal{U} \nrightarrow 2y$ are in bijection with the set $2^{\operatorname{Ob}\mathcal{U}}$.
2. Use the case of $\mathcal{C} := 2y$ to show that if retrofunctors $\mathcal{U} \nrightarrow \mathcal{C}$ are always in bijection with objects in $\mathcal{C}$, then $\mathcal{U}$ must have exactly one object.
3. Now use a different category $\mathcal{D}$ to show that if retrofunctors $\mathcal{U} \nrightarrow \mathcal{D}$ are in bijection with objects in $\mathcal{D}$, then $\mathcal{U}$ must have more than one object. Conclude that objects are not reprsentable in $\mathbf{Cat}^\sharp$ the way they are in $\mathbf{Cat}$.
4. Is there a fixed category $\mathcal{V}$ for which retrofunctors $\mathcal{E} \nrightarrow \mathcal{V}$ are in bijection with objects in $\mathcal{E}$ for every category $\mathcal{E}$? If there is, find it; if there isn't, prove there isn't. ◊

Example 7.92. 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 retrofunctors into $\mathbb{R}y^{\mathbb{R}}$?

Let $\mathcal{C}$ be a category and $|\cdot|\colon \mathcal{C} \nrightarrow \mathbb{R}y^{\mathbb{R}}$ a retrofunctor. 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 7.93 (Solution here). How many retrofunctors

$$\boxed{s \xrightarrow{a} t} \nrightarrow \boxed{u \overset{b}{\underset{c}{\rightrightarrows}} v}$$

are there from the walking arrow category $\mathcal{A}$, drawn above on the left, to the walking parallel-arrows category $\mathcal{PA}$, drawn above on the right? ◊

Exercise 7.94 (Solution here).

1. For any category $\mathcal{C}$ with carrier $\mathfrak{c}$, find a category with carrier $\mathfrak{c}y$.
2. Show that your construction is functorial; i.e. assign each retrofunctor $\mathcal{C} \nrightarrow \mathcal{D}$ a retrofunctor $\mathfrak{c}y \nrightarrow \mathfrak{d}y$ in a way that preserves identities and composites.
3. Is your functor a monad on $\mathbf{Cat}^\sharp$, a comonad on $\mathbf{Cat}^\sharp$, both, or neither? ◊

Exercise 7.95 (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 lenses.

1. If f and $f \mathbin{⨟} g$ are each retrofunctors, is g automatically a retrofunctor? If so, sketch a proof; if not, sketch a counterexample.
2. If g and $f \mathbin{⨟} g$ are each retrofunctors, is f automatically a retrofunctor? If so, sketch a proof; if not, sketch a counterexample. ◊

In the next chapter, we will delve deeper into the categorical structure and properties of $\mathbf{Cat}^\sharp$. We'll encounter many more categories and retrofunctors along the way. But first, we'll conclude this chapter with several alternative characterizations of retrofunctors out of state categories.

7.3.3 Equivalent characterizations of retrofunctors from state categories

Fix a category $\mathcal{C}$ throughout with polynomial carrier $\mathfrak{c}$. How can view the data of a retrofunctor from a state category $\mathcal{S}y^{\mathcal{S}} \nrightarrow \mathcal{C}$? This is actually a very natural categorical concept—we'll see some equivalent ways to express this data below, then state and prove even more equivalences next chapter when we have the machinery to do so.

As coalgebras

Recall from Example 6.67 that for a set S, lenses $Sy^S \to p$ correspond to functions $S \to p(S)$ known as coalgebras for the functor p. As a retrofunctor $Sy^S \nrightarrow \mathscr{C}$ is just a special kind of lens from Sy^S to $\mathfrak{c}$, it should correspond to a special kind of coalgebra $S \to \mathfrak{c}(S)$ for the functor $\mathfrak{c}$. Indeed, whenever $\mathfrak{c}$ carries a comonoid $\mathscr{C}$ with respect to the composition product (i.e. a comonad), there is a special notion of a $\mathscr{C}$-coalgebra (i.e. a coalgebra for the comonad $\mathscr{C}$), as follows.

Definition 7.96 (Coalgebra for a polynomial comonoid)**.** Let $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ be a polynomial comonoid. A *$\mathscr{C}$-coalgebra* (S, α) is

- a set S, called the *carrier*, equipped with
- a function $\alpha\colon S \to \mathfrak{c} \triangleleft S$,

such that the following diagrams, collectively known as the *coalgebra laws*, commute:

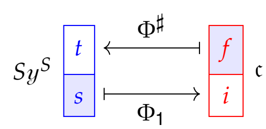
$$
\begin{tikzcd}
S \arrow[r, "\alpha"] \arrow[dr, equal] & \mathfrak{c} \triangleleft S \arrow[d, "\epsilon \triangleleft S"] \\
& S
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
S \arrow[r, "\alpha"] \arrow[d, "\alpha"'] & \mathfrak{c} \triangleleft S \arrow[d, "\delta \triangleleft S"] \\
\mathfrak{c} \triangleleft S \arrow[r, "\mathfrak{c} \triangleleft \alpha"'] & \mathfrak{c} \triangleleft \mathfrak{c} \triangleleft S.
\end{tikzcd}
\tag{7.97}
$$

A *morphism* of $\mathscr{C}$-coalgebras $(S, \alpha) \to (T, \beta)$ is a function $h\colon S \to T$ such that the following diagram commutes:

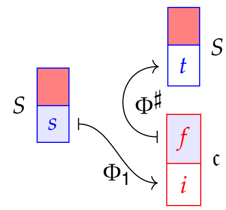
$$
\begin{tikzcd}
S \arrow[r, "\alpha"] \arrow[d, "h"'] & \mathfrak{c} \triangleleft S \arrow[d, "\mathfrak{c} \triangleleft h"] \\
T \arrow[r, "\beta"'] & \mathfrak{c} \triangleleft T
\end{tikzcd}
$$

Proposition 7.98. Retrofunctors $Sy^S \nrightarrow \mathscr{C}$ can be identified (up to isomorphism) with $\mathscr{C}$-coalgebras carried by S.

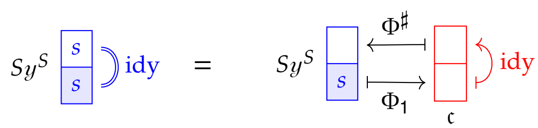
Proof. Let $\mathfrak{c}$ be the carrier of $\mathscr{C}$. In Example 6.67, we showed that (6.65) gives a natural correspondence between lenses $\Phi\colon Sy^S \to \mathfrak{c}$ and functions $\varphi\colon S \to \mathfrak{c} \triangleleft S$. We can unravel this correspondence via the proof of Proposition 6.57 as follows. A lens $\Phi\colon Sy^S \to \mathfrak{c}$ can be drawn like so (we will adopt our former convention of identifying each morphism $s \to t$ from Sy^S with its codomain t):



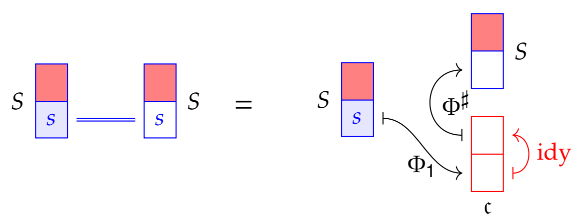
Meanwhile, the corresponding function $\varphi\colon S \to \mathfrak{c} \triangleleft S$, equivalently a lens between constants, can be drawn thusly (recall that we color a box red when it is impossible to fill, i.e. when it can only be filled by an element of the empty set):



Then it suffices to show that Φ satisfies the retrofunctor laws if and only if φ satisfies the coalgebra laws. We can verify this using polyboxes. From (7.53), the eraser preservation law for Φ would state the following (remember that the arrow in the eraser for the state category Sy^S is just an equality):

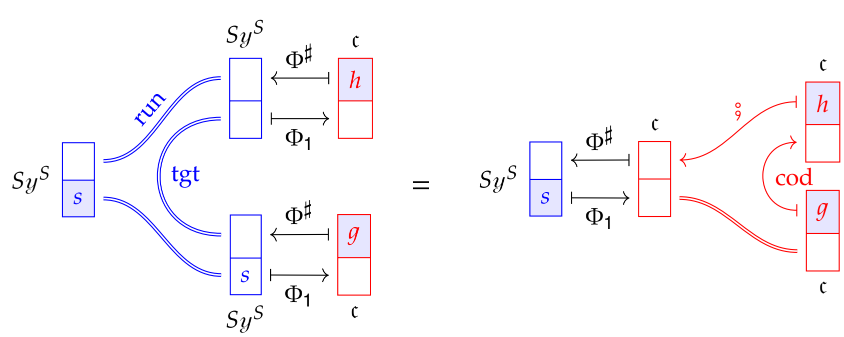


Meanwhile, the commutative triangle on the left of (7.97) can be written as follows:

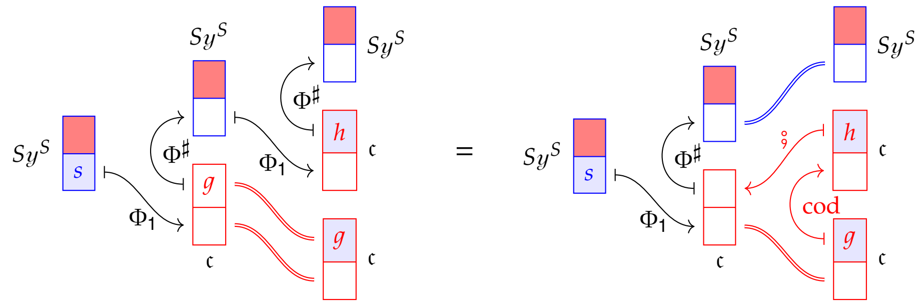


But these polybox equations are entirely equivalent.

Then from (7.54), the duplicator preservation law for Φ would state the following (remember that the three arrows in the duplicator for the state category Sy^S are all equalities)



Meanwhile, the commutative triangle on the right of (7.97) can be written as follows:



But these polybox equations are equivalent as well. Hence the retrofunctor laws for Φ are equivalent to the coalgebra laws for φ. $\square$

So a retrofunctor from a state category to $\mathcal{C}$ bears the same data as a $\mathcal{C}$-coalgebra. The equivalences don't stop there, however.

As discrete opfibrations

The concept of a $\mathcal{C}$-coalgebra is in turn equivalent to a better known categorical construction on $\mathcal{C}$, which we introduce here.

Definition 7.99 (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 $\overline{f}\colon s \to s'$ such that $\pi(\overline{f}) = f$. Note that in this case $\pi(s') = c$.

$$
\begin{tikzcd}
s \arrow[r, dashed, "\overline{f}"] \arrow[d, mapsto, "\pi"'] & s' \arrow[d, mapsto, "\pi"] \\
\pi(s) \arrow[r, "f"'] & \_
\end{tikzcd}
$$

A *morphism* of discrete opfibrations $(\mathcal{S}, \pi) \to (\mathcal{S}', \pi')$ over $\mathcal{C}$ is a functor $F\colon \mathcal{S} \to \mathcal{S}'$ making the following triangle commute: We refer to $\overline{f}$ as the *lift* of f to s.

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:

$$
\begin{tikzcd}
\mathcal{S} \arrow[rr, "F"] \arrow[dr, "\pi"'] & & \mathcal{S}' \arrow[dl, "\pi'"] \\
& \mathcal{C} &
\end{tikzcd}
\tag{7.100}
$$

We denote the category of discrete opfibrations over $\mathcal{C}$ by $\mathbf{dopf}(\mathcal{C})$.

Exercise 7.101 (Solution here). Show that if $F\colon \mathcal{S} \to \mathcal{S}'$ is a functor making the triangle (7.100) commute, where both π and π' are discrete opfibrations, then F is also a discrete opfibration. ◊

Exercise 7.102 (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 7.99, 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 $\overline{f} \mathbin{⨾} \overline{g} = \overline{f \mathbin{⨾} g}$.
3. Show how π could instead be interpreted as a retrofunctor. ◊

As it turns out, not only do $\mathcal{C}$-coalgebras carry the same data as discrete opfibrations over $\mathcal{C}$, they in fact comprise isomorphic categories.

Proposition 7.103. The category of $\mathcal{C}$-coalgebras is isomorphic to the category $\mathbf{dopf}(\mathcal{C})$ of discrete opfibrations over $\mathcal{C}$.

As copresheaves

It is well-known in the category theory literature that the category of discrete opfibrations over $\mathcal{C}$ is equivalent to yet another familiar category: the category $\mathbf{Set}^{\mathcal{C}}$, whose objects are functors $\mathcal{C} \to \mathbf{Set}$. Such a functor is known as a *copresheaf* on $\mathcal{C}$ for short. These are relevant, e.g. in the theory of categorical databases [Spi12]. Here we review what is needed to understand this equivalence. We begin by giving a standard construction on any copresheaf.

Definition 7.104 (Category of elements)**.** Given a copresheaf on $\mathcal{C}$, i.e. a functor $I\colon \mathcal{C} \to \mathbf{Set}$, its *category of elements* $\int^{\mathcal{C}} I$ is defined to have objects

$$\mathrm{Ob} \int^{\mathcal{C}} I \coloneqq \{(c, x) \mid c \in \mathcal{C}, x \in Ic\}$$

and a morphism $f\colon (c, x) \to (c', x')$ for every morphism $f\colon c \to c'$ from $\mathcal{C}$ satisfying

$$(If)(x) = x'.$$

Identities and composites in $\int^{\mathcal{C}} I$ are inherited from $\mathcal{C}$; they obey the usual category laws by the functoriality of I.

The category is so named because its objects are the elements of the sets that the objects of $\mathcal{C}$ are sent to by I. Each morphism $f\colon c \to \_$ in $\mathcal{C}$ then becomes as many morphisms in $\int^{\mathcal{C}} I$ as there are elements of Ic, tracking where If sends each such element.

The next exercise shows how this construction turns every copresheaf into a discrete opfibration.

Exercise 7.105 (Solution here). Let $I\colon \mathcal{C} \to \mathbf{Set}$ be a functor, and let $\int^{\mathcal{C}} I$ be as in Definition 7.104.

1. Show that there is a functor $\pi\colon \int^{\mathcal{C}} I \to \mathcal{C}$ sending objects $(c, x) \mapsto c$ and morphisms $f : (c, x) \to (c', x')$ to $f : c \to c'$.
2. Show that π is in fact a discrete opfibration. ◊

In fact, the assignment of a discrete opfibration to every copresheaf given above is functorial, as the next exercise shows.

Exercise 7.106 (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 7.99.
3. Have you now verified that there is a functor

$$\int^{\mathcal{C}} : \mathbf{Set}^{\mathcal{C}} \to \mathbf{dopf}(\mathcal{C})$$

or is there something left to do?

◊

Exercise 7.107 (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 7.108. The category $\mathbf{Set}^{\mathcal{C}}$ of copresheaves on $\mathcal{C}$ is equivalent to the category of discrete opfibrations over $\mathcal{C}$.

Proof. By Exercise 7.106 we have a functor $\int^{\mathcal{C}} : \mathbf{Set}^{\mathcal{C}} \to \mathbf{dopf}(\mathcal{C})$. There is a functor going back: given a discrete opfibration $\pi\colon \mathcal{S} \to \mathcal{C}$, we define a functor $\partial\pi\colon \mathcal{C} \to \mathbf{Set}$ on objects by sending each $c \in \mathcal{C}$ to the set of objects in $\mathcal{S}$ that π maps to c; that is,

$$(\partial\pi)(c) \coloneqq \{s \in \mathcal{S} \mid \pi(s) = c\}.$$

Then on morphisms, for each $f\colon c \to \_$ in $\mathcal{C}$ and $s \in (\partial\pi)(c)$ we have $\pi(s) = c$, so by Definition 7.99 there exists a unique morphism $\overline{f}\colon s \to \_$ for which $\pi(\overline{f}) = f$. As $\pi(\operatorname{cod}\overline{f}) = \operatorname{cod} f$, we have $\operatorname{cod}\overline{f} \in (\partial\pi)(\operatorname{cod} f)$, so we can define

$$(\partial\pi)(f)(s) \coloneqq \operatorname{cod}\overline{f}.$$

On objects, the roundtrip $\mathbf{Set}^{\mathcal{C}} \to \mathbf{Set}^{\mathcal{C}}$ sends $I\colon \mathcal{C} \to \mathbf{Set}$ to the functor

$$\begin{aligned} c \mapsto \{s \in \textstyle\int^{\mathcal{C}} I \mid \pi(s) \\ = \{(c,x) \mid x \in I(c)\} && = I(c). \end{aligned}$$

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 $\mathcal{S} \cong \int^{\mathcal{C}} \partial\pi$. □

Proposition 7.109. 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.

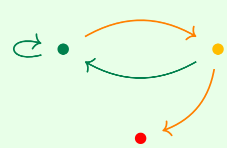
Proof. Given a discrete opfibration $\pi\colon \mathcal{S} \to \mathcal{C}$, take $S \coloneqq \mathrm{Ob}(\mathcal{S})$ and define $(\varphi_1, \varphi^\sharp)\colon Sy^S \to \mathfrak{c}$ by $\varphi_1 = \pi$ and with $\varphi^\sharp$ given by the lifting: $\varphi(g) \coloneqq \hat{g}$ as in Definition 7.99. One checks using Exercise 7.102 that this defines a retrofunctor.

Conversely, given a retrofunctor $(\varphi_1, \varphi^\sharp)\colon Sy^S \to \mathfrak{c}$, the function φ_1 induces a lens $Sy \to \mathfrak{c}$, 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 retrofunctor; see Exercise 7.110. □

Exercise 7.110 (Solution here). With notation as in Proposition 7.109, complete the proof as follows.

1. Check that $(\varphi, \varphi^\sharp)$ defined in the first paragraph is indeed a retrofunctor.
2. Find a comonoid structure on $\mathfrak{s}$ such that ψ is a retrofunctor, as stated in the second paragraph.
3. Show that the two directions are inverse, up to isomorphism. ◊

Example 7.111. In Example 8.50 we had a dynamical system with $S \coloneqq \{\bullet, \bullet, \bullet\}$ and $p \coloneqq y^2 + 1$, and $\varphi\colon Sy^S \to p$ from Exercise 4.23, depicted here again for your convenience:

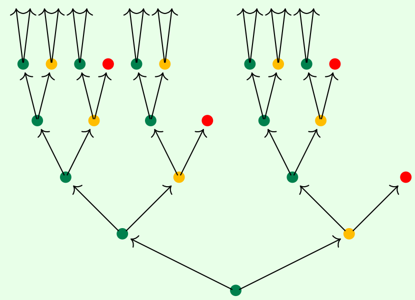


(7.112)

Under the forgetful-cofree adjunction (Theorem 8.45), the lens φ coincides with a retrofunctor $F\colon Sy^S \nrightarrow \mathscr{T}_p$ from the state category on S to the category of p-trees. We can now see this as a copresheaf on the category $\mathscr{T}_p$ itself.

The cofree category $\mathscr{T}_t$ is actually the free category on a graph, as we saw in Proposition 8.55, 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 retrofunctor $\varphi\colon Sy^S \nrightarrow \mathscr{T}_p$. By Propositions 7.108 and 7.109, we can consider it as a discrete opfibration over $\mathscr{T}_p$. By Exercise 7.107 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 (7.112).

To summarize, we have four equivalent notions:

(1) retrofunctors $F\colon Sy^S \nrightarrow \mathcal{C}$;
(2) $\mathcal{C}$-coalgebras (S, α), with $\alpha\colon S \to \mathfrak{c} \triangleleft S$;
(3) discrete opfibrations $\pi\colon \mathcal{S} \to \mathcal{C}$, with $\mathrm{Ob}\,\mathcal{S} = S$;
(4) copresheaves $I\colon \mathcal{C} \to \mathbf{Set}$, with $\mathrm{Ob} \int^{\mathcal{C}} I = S$.

Moreover, (2), (3), and (4) form equivalent categories. Translating between these notions yields different perspectives on familiar categorical concepts. In the next chapter, we will discover even more characterizations of the same data within **Poly** (see Proposition 8.66).

7.4 Summary and further reading

In this chapter we began by showing that for every set S, the thing that makes a dynamical system like $Sy^S \to p$ actually run is the fact that Sy^S has the structure of a comonoid. We then explained Ahman-and-Uustalu's result that comonoids, i.e. polynomials p, equipped with a pair of lenses $\epsilon\colon p \to y$ and $\delta\colon p \to p \triangleleft p$, are exactly categories [AU16]. We explained how ϵ picks out an identity for each object and how δ picks out a codomain for each morphism and a composite for each composable pair of morphisms. In particular we showed that the category corresponding to Sy^S is the *contractible groupoid on* S, i.e. the category with S-many objects and a unique morphism between any two.

We then discussed how comonoid morphisms $\mathfrak{c} \to \mathfrak{d}$ are not functors but *retrofunctors*: they map forwards on objects and backwards on morphisms. Retrofunctors were first defined by Marcelo Aguiar [Agu97], though his definition was opposite to ours; he referred to these as *cofunctors*. Retrofunctors are the morphisms of a category we notate as $\mathbf{Cat}^\sharp$. We showed that retrofunctors between state categories $Sy^S \nrightarrow Ty^T$ are what are known in the functional programming community as *very well-behaved lenses*. For more on lenses, see [nLa22].

7.5 Exercise solutions

Solution to Exercise 7.2.

Given a polynomial $\mathfrak{s} \in \mathbf{Poly}$ equipped with a lens $\epsilon\colon \mathfrak{s} \to y$, we know that ϵ picks out a direction at every position of $\mathfrak{s}$. So all we can say about $\mathfrak{s}$ is that there is at least one direction at each of its positions. Equivalently, we could say that $\mathfrak{s}$ can be written as the product of y and some other polynomial.

Solution to Exercise 7.9.

Following the arrows on either side of (7.8) all the way to the domain's direction box, we obtain an expression for each box's contents that we can then set equal to each other. The easiest way to actually write down these expressions is probably to start at the end with $s_0 \to s_3$ and follow the arrows backward, unpacking each term until only the contents of the blue boxes remain (namely $s_0, s_0 \to s_1, s_1 \to s_2$, and $s_2 \to s_3$). Here's what we get when we follow this process for the left hand side of (7.8) (remember where to look for the three inputs to the run function):

$$\begin{aligned} s_0 \to s_3 &= \mathrm{run}(s_0, s_0 \to s_2, s_2 \to s_3) \\ &= \mathrm{run}(s_0, \mathrm{run}(s_0, s_0 \to s_1, s_1 \to s_2), s_2 \to s_3) \\ &= \mathrm{run}(s_0, \mathrm{run}(s_0, s_0 \to s_1, s_1 \to s_2), s_2 \to s_3); \end{aligned}$$

and here's what we get for the right (also remember where to look for the two inputs to the target function):

$$\begin{aligned} s_0 \to s_3 &= \mathrm{run}(s_0, s_0 \to s_1, s_1 \to s_3) \\ &= \mathrm{run}(s_0, s_0 \to s_1, \mathrm{run}(s_1, s_1 \to s_2, s_2 \to s_3)) \\ &= \mathrm{run}(s_0, s_0 \to s_1, \mathrm{run}(\mathrm{tgt}(s_0, s_0 \to s_1), s_1 \to s_2, s_2 \to s_3)). \end{aligned}$$

Setting these equal yields our desired associativity equation:

$$\mathrm{run}(s_0, \mathrm{run}(s_0, s_0 \to s_1, s_1 \to s_2), s_2 \to s_3) = \mathrm{run}(s_0, s_0 \to s_1, \mathrm{run}(\mathrm{tgt}(s_0, s_0 \to s_1), s_1 \to s_2, s_2 \to s_3)).$$

Solution to Exercise 7.11.

1. Given $\mathfrak{s} \in \mathbf{Poly}$ and a lens $\delta\colon \mathfrak{s} \to \mathfrak{s} \triangleleft \mathfrak{s}$, we want all the ways to obtain a lens $\mathfrak{s} \to \mathfrak{s}^{\triangleleft 4}$ using $\delta, \mathrm{id}_{\mathfrak{s}}$ (i.e. $\mathfrak{s}$), $\triangleleft$, and $\fatsemi$. Starting with $\mathfrak{s}$, the only way to get to $\mathfrak{s}^{\triangleleft 2}$ is with a single $\delta\colon \mathfrak{s} \to \mathfrak{s}^{\triangleleft 2}$. From there, we can get to $\mathfrak{s}^{\triangleleft 4}$ directly by composing with $\delta \triangleleft \delta$ to obtain $\delta \fatsemi (\delta \triangleleft \delta)$. Alternatively, we can preserve either the first or the second $\mathfrak{s}$ using the identity, then get to $\mathfrak{s}^{\triangleleft 3}$ from the other $\mathfrak{s}$ in one of two ways: either $\delta \fatsemi (\delta \triangleleft \mathfrak{s})$ or $\delta \fatsemi (\mathfrak{s} \triangleleft \delta)$. This gives us 4 more ways to write a lens $\mathfrak{s} \to \mathfrak{s}^{\triangleleft 4}$: either $\delta \fatsemi (\mathfrak{s} \triangleleft (\delta \fatsemi (\delta \triangleleft \mathfrak{s})))$ or $\delta \fatsemi (\mathfrak{s} \triangleleft (\delta \fatsemi (\mathfrak{s} \triangleleft \delta)))$ if we chose to preserve the first $\mathfrak{s}$, and either $\delta \fatsemi ((\delta \fatsemi (\delta \triangleleft \mathfrak{s})) \triangleleft \mathfrak{s})$ or $\delta \fatsemi ((\delta \fatsemi (\mathfrak{s} \triangleleft \delta)) \triangleleft \mathfrak{s})$ if we chose to preserve the second. Here's the full list, sorted roughly by how far to the left we try to apply each δ:
   (1) $\delta \fatsemi ((\delta \fatsemi (\delta \triangleleft \mathfrak{s})) \triangleleft \mathfrak{s})$
   (2) $\delta \fatsemi ((\delta \fatsemi (\mathfrak{s} \triangleleft \delta)) \triangleleft \mathfrak{s})$
   (3) $\delta \fatsemi (\delta \triangleleft \delta)$

(4) $\delta \mathbin{⨟} (\mathfrak{s} \triangleleft (\delta \mathbin{⨟} (\delta \triangleleft \mathfrak{s})))$
(5) $\delta \mathbin{⨟} (\mathfrak{s} \triangleleft (\delta \mathbin{⨟} (\mathfrak{s} \triangleleft \delta)))$

This coincides with the 5 different ways to parenthesize a 4-term expression.

2. We wish to show that if (7.10) commutes, then all the lenses on our list are equal. The commutativity of (7.10) implies that $\delta \mathbin{⨟} (\delta \triangleleft \mathfrak{s}) = \delta \mathbin{⨟} (\mathfrak{s} \triangleleft \delta)$; so (1) and (2) from our list are equal, as are (4) and (5). Meanwhile, since $\mathfrak{s} = \mathfrak{s} \mathbin{⨟} \mathfrak{s}$, we can rewrite (1) as

$$\delta \mathbin{⨟} ((\delta \mathbin{⨟} (\delta \triangleleft \mathfrak{s})) \triangleleft (\mathfrak{s} \mathbin{⨟} \mathfrak{s})) = \delta \mathbin{⨟} (\delta \triangleleft \mathfrak{s}) \mathbin{⨟} (\delta \triangleleft \mathfrak{s} \triangleleft \mathfrak{s}),$$

where the associativity of $\triangleleft$ and $⨟$ allows us to drop some parentheses. Then the commutativity of (7.10) allows us to further rewrite this as

$$\begin{aligned}\delta \mathbin{⨟} (\mathfrak{s} \triangleleft \delta) \mathbin{⨟} (\delta \triangleleft \mathfrak{s} \triangleleft \mathfrak{s}) &= \delta \mathbin{⨟} ((\mathfrak{s} \mathbin{⨟} \delta) \triangleleft (\delta \mathbin{⨟} (\mathfrak{s} \triangleleft \mathfrak{s})))\\ &= \delta \mathbin{⨟} (\delta \triangleleft \delta),\end{aligned}$$

so (1) and (3) are equal. Similarly, we can rewrite (5) as

$$\begin{aligned}\delta \mathbin{⨟} ((\mathfrak{s} \mathbin{⨟} \mathfrak{s}) \triangleleft (\delta \mathbin{⨟} (\mathfrak{s} \triangleleft \delta))) &= \delta \mathbin{⨟} (\mathfrak{s} \triangleleft \delta) \mathbin{⨟} (\mathfrak{s} \triangleleft \mathfrak{s} \triangleleft \delta)\\ &= \delta \mathbin{⨟} (\delta \triangleleft \mathfrak{s}) \mathbin{⨟} (\mathfrak{s} \triangleleft \mathfrak{s} \triangleleft \delta)\\ &= \delta \mathbin{⨟} ((\delta \mathbin{⨟} (\mathfrak{s} \triangleleft \mathfrak{s})) \triangleleft (\mathfrak{s} \mathbin{⨟} \delta))\\ &= \delta \mathbin{⨟} (\delta \triangleleft \delta)\end{aligned}$$

so (3) and (5) are equal. Hence all the lenses on our list are equal.

Solution to Exercise 7.12.

We have $\mathrm{Run}_n(\varphi) = \delta^{(n)} \mathbin{⨟} \varphi^{\triangleleft n}$ for all $n \in \mathbb{N}$, as well as $\delta^{(0)} = \epsilon$ and $\delta^{(1)} = \mathrm{id}_{\mathfrak{s}}$. Then $\mathrm{Run}_0(\varphi) = \epsilon \mathbin{⨟} \varphi^{\triangleleft 0} = \epsilon \mathbin{⨟} \mathrm{id}_y = \epsilon$ and $\mathrm{Run}_1(\varphi) = \mathrm{id}_{\mathfrak{s}} \mathbin{⨟} \varphi^{\triangleleft 1} = \varphi^{\triangleleft 1} = \varphi$.

Solution to Exercise 7.21.

We complete the proof of Proposition 7.20, where we are given a comonoid $(\mathfrak{c}, \epsilon, \delta)$ along with $\delta^{(0)} := \epsilon$ and $\delta^{(n+1)} := \delta \mathbin{⨟} \left(\delta^{(n)} \triangleleft \mathfrak{c}\right)$ for all $n \in \mathbb{N}$.

1. We will show that $\delta^{(n)}$ is a morphism $\mathfrak{c} \to \mathfrak{c}^{\triangleleft n}$ for every $n \in \mathbb{N}$ by induction on n. We know $\delta^{(0)} = \epsilon$ is a morphism $\mathfrak{c} \to y = \mathfrak{c}^{\triangleleft 0}$, and for each $n \in \mathbb{N}$, if $\delta^{(n)}$ is a morphism $\mathfrak{c} \to \mathfrak{c}^{\triangleleft n}$, then the composite $\delta^{(n+1)} = \delta \mathbin{⨟} \left(\delta^{(n)} \triangleleft \mathfrak{c}\right)$ is a morphism

$$\mathfrak{c} \xrightarrow{\delta} \mathfrak{c} \triangleleft \mathfrak{c} \xrightarrow{\delta^{(n)} \triangleleft \mathfrak{c}} \mathfrak{c}^{\triangleleft n} \triangleleft \mathfrak{c} \cong \mathfrak{c}^{\triangleleft (n+1)}$$

Hence the result follows by induction.

2. We have $\delta^{(1)} = \delta \mathbin{⨟} (\delta^{(0)} \triangleleft \mathfrak{c}) = \delta \mathbin{⨟} (\epsilon \triangleleft \mathfrak{c}) = \mathrm{id}_{\mathfrak{c}}$ by the left erasure law from (7.15).
3. By the previous part, we have $\delta^{(2)} = \delta \mathbin{⨟} (\delta^{(1)} \triangleleft \mathfrak{c}) = \delta \mathbin{⨟} (\mathfrak{c} \triangleleft \mathfrak{c}) = \delta$.

Solution to Exercise 7.26.

We will eventually see that comonoids in **Poly** are categories; this gives us intuition for what is going on here. In fact the category corresponding to the comonoid in this exercise is the walking arrow, depicted in (7.24). The erasure (counit) laws are verified by looking at the picture of δ in (7.25) and noting that it sends the double (do-nothing) lines (the identities) back to the double lines.

Solution to Exercise 7.27.

Given a set B, we wish to give a comonoid structure on By and show that it is unique. There is only one way to define an eraser lens $\epsilon\colon By \to y$: the on-position function is the unique map $!\colon B \to 1$, while every on-directions function $1 \to 1$ must be the identity. Meanwhile $By \triangleleft By \cong B^2y$, so to specify a duplicator lens $\delta\colon By \to B^2y$, it suffices to specify an on-positions function $\delta_1\colon B \to B^2$, and every on-directions function will again be the identity.

All the comonoid laws should then hold trivially on directions, so it suffices to consider each law on positions. The erasure laws imply that the composite functions

$$B \xrightarrow{\delta_1} B \times B \xrightarrow{B \times !} B \times 1 \cong B,$$

where the second map is just the canonical left projection, and

$$B \xrightarrow{\delta_1} B \times B \xrightarrow{! \times B} 1 \times B \cong B,$$

where the second nap is just the canonical right projection, are both the identity on B. The only function $\delta_1 \colon B \to B \times B$ that satisfies this condition is the diagonal $a \mapsto (a, a)$. It is easy to verify that the diagonal is coassociative, so this does define a unique comonoid structure on By. (It turns out that this is equivalent to the well-known result that there is a unique comonoid structure on every set in $(\mathbf{Set}, 1, \times)$.)

Solution to Exercise 7.33.

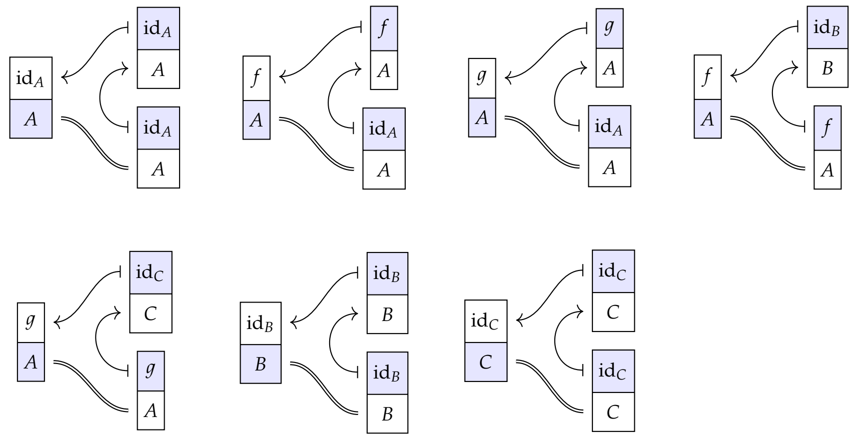
1. The category $\boxed{A \xrightarrow{f} B}$ has 2 morphisms out of A, namely id_A and f; and 1 morphism out of B, namely id_B. So its carrier is $\{A\}y^{\{\mathrm{id}_A, f\}} + \{B\}y^{\{\mathrm{id}_B\}} \cong y^2 + y$.
2. The category $\boxed{B \xrightarrow{g} A \xleftarrow{h} C}$ has 1 morphism out of A, namely id_A; 2 morphisms out of B, namely id_B and g; and 2 morphisms out of C, namely id_C and h. So its carrier is $\{A\}y^{\{\mathrm{id}_A\}} + \{B\}y^{\{\mathrm{id}_B, g\}} + \{C\}y^{\{\mathrm{id}_C, h\}} \cong 2y^2 + y$.
3. The empty category has no objects or morphisms, so its carrier is just 0.
4. The category in question has 1 object, and its set of morphisms is in bijection with $\mathbb{N}$, so its carrier is isomorphic to $y^{\mathbb{N}}$. (This category is the monoid $(\mathbb{N}, 0, +)$ viewed as a 1-object category; see Example 7.40 for the general case.)
5. The category in question has $\mathbb{N}$ as its set of objects, and for each $m \in \mathbb{N}$, the morphisms out of m are determined by their codomains: there is exactly 1 morphism $m \to n$ for every $n \in \mathbb{N}$ satisfying $m \leq n$, and no other morphisms out of m. So the carrier of the category is isomorphic to
$$\sum_{m \in \mathbb{N}} y^{\{n \in \mathbb{N} \mid m \leq n\}} \cong \mathbb{N}y^{\mathbb{N}},$$
as $\{n \in \mathbb{N} \mid m \leq n\} \cong \mathbb{N}$ under the bijection $n \mapsto n - m$. (This category is the poset $(\mathbb{N}, \leq)$.)
6. The category in question has $\mathbb{N}$ as its set of objects, and for each $n \in \mathbb{N}$, the morphisms out of n are again determined by their codomains: there is exactly 1 morphism $n \to m$ for every $m \in \mathbb{N}$ satisfying $m \leq n$, and no other morphisms out of n. So the carrier of the category is isomorphic to
$$\sum_{n \in \mathbb{N}} y^{\{m \in \mathbb{N} \mid m \leq n\}} \cong \sum_{n \in \mathbb{N}} y^{\mathsf{n}+1} \cong y^1 + y^2 + y^3 + \cdots.$$
(This category is the poset $(\mathbb{N}, \geq)$.)

Solution to Exercise 7.35.

We are given a comonoid $(\mathfrak{c}, \epsilon, \delta)$ corresponding to the preorder $\boxed{B \xleftarrow{f} A \xrightarrow{g} C}$.

1. There are three morphisms with domain A, namely id_A, f, and g; the only other morphisms are the identity morphisms on B and C. So the carrier is $\mathfrak{c} = \{A\}y^{\{\mathrm{id}_A, f, g\}} + \{B\}y^{\{\mathrm{id}_B\}} + \{C\}y^{\{\mathrm{id}_C\}}$.
2. It suffices to specify the eraser $\epsilon \colon \mathfrak{c} \to y$ on directions: as always, $\epsilon_i^\sharp \colon 1 \to \mathfrak{c}[i]$ picks out id_i for each $i \in \mathfrak{c}(1) = \{A, B, C\}$.
3. The duplicator $\delta \colon \mathfrak{c} \to \mathfrak{c} \triangleleft \mathfrak{c}$ tells us the codomain of each morphism, as well as how every pair of composable morphisms compose (which in the case of a preorder can be deduced automatically).

So we can completely characterize the behavior of δ using polyboxes as follows:



Solution to Exercise 7.36.

The linear polynomial By corresponds to a category whose objects form the set B and whose only morphisms are identities: in other words, it is the discrete category on B.

Solution to Exercise 7.37.

1. The polynomial $p := y^{n+1} + \mathsf{n}y$ has $n+1$ positions: 1 with $n+1$ directions and the rest with 1 direction each. So any category carried by p has $n+1$ objects, of which only 1 has any nonidentity morphisms coming out of it: in fact, it has n nonidentity morphisms coming out of it. But if the category is to be a preorder, each of these n nonidentity morphisms must have a distinct codomain. As there are exactly n other objects, this completely characterizes the category. Equivalently, it is the discrete category on n adjoined with a unique initial object, so that the only nonidentity morphisms are the morphisms out of that initial object to each of the other objects exactly once.
2. This category can be thought of as "star-shaped" if we picture the initial object in the center with morphisms leading out to the other n objects like spokes.

Solution to Exercise 7.39.

In the case of $S := 0$, the only comonoid structure on $Sy^S \cong 0$ is given by the empty category, the only category with no objects; and in the case of $S := 1$, the only comonoid structure on $Sy^S \cong y$ is given by the category with 1 object and no nonidentity morphisms, again the only such category. So in those cases, the comonoid structure on Sy^S is unique.

Now assume $|S| \geq 2$. The state category is always connected, but we can always find a comonoid structure on Sy^S given by a category that is not connected—and thus not isomorphic to the state category—as follows. Consider a category whose object set is S that has no morphisms between distinct objects, so that it is certainly not connected. Then to specify the category, it suffices to specify a monoid associated with each object that will give the morphisms whose domain and codomain are equal to that object. But there is always a monoid structure on a given nonempty set S. If S is finite, we can take the cyclic group $\mathbb{Z}/|S|\mathbb{Z}$ of order $|S|$, so that the resulting category has carrier $Sy^{\mathbb{Z}/|S|\mathbb{Z}} \cong Sy^S$. On the other hand, if S is infinite, we can take the free monoid on S, which has cardinality $\sum_{n \in \mathbb{N}} |S|^{\mathsf{n}} = |\mathbb{N}||S| = |S|$. So the resulting category will again have carrier Sy^S.

Solution to Exercise 7.41.

The fact that monoids $(M, e, *)$ in $(\mathbf{Set}, 1, \times)$ are just comonoids (y^M, ϵ, δ) in $(\mathbf{Poly}, y, \triangleleft)$, following the construction of Example 7.40, is a direct consequence of the fully faithful Yoneda embedding $\mathbf{Set}^{\mathrm{op}} \to \mathbf{Poly}$ sending $A \mapsto y^A$ that maps $1 \mapsto y$, $A \times B \mapsto y^{A \times B} \cong y^A \triangleleft y^B$ naturally, and $M \mapsto y^M$.

We can also state the laws and the correspondences between them explicitly, keeping in mind that e and $*$ are just the on-directions functinos of ϵ and δ. The monoid's unitality condition states that e is a 2-sided unit for $*$, or that

$$\begin{tikzcd}
1 \times M \arrow[r, equal] \arrow[dr, "e \times M"'] & M & M \times 1 \arrow[l, equal] \arrow[dl, "M \times e"] \\
& M \times M, \arrow[u, "*"] &
\end{tikzcd}$$

commutes—equivalent to the comonoid's erasure laws, which state that

$$\begin{tikzcd}
y \triangleleft y^M \arrow[r, equal] & y^M \arrow[d, "\delta"] & y^M \triangleleft y \arrow[l, equal] \\
& y^M \triangleleft y^M, \arrow[ul, "\epsilon \triangleleft y^M"] \arrow[ur, "y^M \triangleleft \epsilon"'] &
\end{tikzcd}$$

commutes (trivial on positions, equivalent to the monoid's unitality condition on directions). Similarly, the monoid's associativity condition states that $*$ is associative, or that

$$\begin{tikzcd}
M & M \times M \arrow[l, "*"'] \\
M \times M \arrow[u, "*"] & M \times M \times M, \arrow[l, "* \times M"] \arrow[u, "M \times *"']
\end{tikzcd}$$

commutes—equivalent to the comonoid's coassociative law, which state that

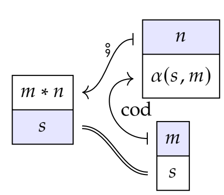
$$\begin{tikzcd}
y^M \arrow[r, "\delta"] \arrow[d, "\delta"'] & y^M \triangleleft y^M \arrow[d, "y^M \triangleleft \delta"] \\
y^M \triangleleft y^M \arrow[r, "\delta \triangleleft y^M"'] & y^M \triangleleft y^M \triangleleft y^M,
\end{tikzcd}$$

commutes (trivial on positions, equivalent to the monoid's associativity condition on directions).

Solution to Exercise 7.44.

Here $(M, e, *)$ is a monoid, S is a set, $\alpha\colon S \times M \to S$ is a monoid action, and $\mathcal{MA}$ is the associated category, whose corresponding comonoid is (Sy^M, ϵ, δ). We also know that for each $s \in S$ and $m \in M$, there is a morphism $s \xrightarrow{m} \alpha(s, m)$ in $\mathcal{MA}$.

1. The erasure $\epsilon\colon Sy^M \to y$ picks out an element of $m \in M$ for every element $s \in S$ that will play the role of an identity, which in particular should also have s as its codomain. Since we want the codomain of the morphism m out of s to be $\alpha(s, m)$, we can take $m = e$ to guarantee that its codomain will be $\alpha(s, e) = s$. So we let ϵ be the lens whose on-directions function at each $s \in S$ is $\epsilon_s^\sharp\colon 1 \to M$ always maps to e.
2. The duplicator $\delta\colon Sy^M \to Sy^M \triangleleft Sy^M$ is determined by what we want the codomain of each morphism to be and how we want the morphisms to compose. We already know that we want the morphism $m \in M$ out of each $s \in S$ to have the codomain $\alpha(s, m)$. If we then have another morphism $n \in M$ out of $\alpha(s, m)$, its codomain will be $\alpha(\alpha(s, m), n) = \alpha(s, m * n)$, the same as the codomain of the morphism $m * n$ out of s. So it makes sense for the composite $s \xrightarrow{m} \alpha(s, m) \xrightarrow{n} \alpha(\alpha(s, m), n)$ to be the morphism $s \xrightarrow{m*n} \alpha(s, m * n)$. Thus, we can define δ in polyboxes as



3. We constructed δ above so that its bottom arrow is an identity function, so verifying the erasure laws amounts to checking that the direction $e \in M$ that ϵ picks out at each position $s \in S$ really does function as an identity morphism $s \xrightarrow{e} \alpha(s, e)$ under the codomain and composition operations specified by δ. We have already ensured that the codomain of e at s is $\alpha(s, e) = s$; meanwhile, given $m \in M$ we have that the composite of $s \xrightarrow{m} \alpha(s, m) \xrightarrow{e} \alpha(s, m)$ is $m * e = m$ and that the composite of $s \xrightarrow{e} s \xrightarrow{m} \alpha(s, m)$ is $e * m = m$ by the monoid's own unit laws. So the erasure laws hold.
4. Verifying the coassociativity of δ amounts to checking that composition plays nicely with codomains and is associative. We already checked the former when defining δ, and the latter follows from the monoid's own associativity laws: given $m, n, p \in M$, we have $(m*n)*p = m*(n*p)$.
5. The associated category $\mathcal{MA}$ is a category whose objects are the elements of the set S being acted on, and whose morphisms $s \to t$ for each $s, t \in S$ are the elements of the monoid $m \in M$ that send s to t, i.e. $\alpha(s, m) = t$. The identity morphism on each object is just the unit $e \in M$, while morphisms compose via the multiplication $*$.
6. It is the same iff M is a group, i.e. if every $m \in M$ has an inverse. Indeed, the comonoid structure on My^M from Example 7.38 corresponds to a category in which every map is an isomorphism, so for each $m \in M$, the left-action $\alpha(m, -)\colon M \to M$ would need to be a bijection, and this is the case iff M is a group.

Solution to Exercise 7.46.

1. Since $\mathbb{R}$ acts on $\mathbb{R}/\mathbb{Z}$ by addition modulo 1, (e.g. $\alpha(.7, 5.4) = .1$), we obtain a comonoid structure on $(\mathbb{R}/\mathbb{Z})y^{\mathbb{R}}$ by Example 7.43. For example, the erasure $(\mathbb{R}/\mathbb{Z})y^{\mathbb{R}} \to y$ sends everything to $0 \in \mathbb{R}$, because 0 is the identity in $\mathbb{R}$.
2. Yes it is a groupoid because $\mathbb{R}$ is a group: every element is invertible.

Solution to Exercise 7.48.

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 7.33 #4), the poset $(\mathbb{N}, \leq)$ (see Exercise 7.33 #5), and the state category on $\mathbb{N}$ (see Example 7.38).
4. Up to isomorphism, there are 3 categories with just one linear and one quadratic object. They can be distinguished by the behavior of the single nonidentity 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. So there are 3 possible categories in total.
5. Yes: since (isomorphic) categories correspond to (isomorphic) comonoids, there are as many categories with one linear and one quadratic object up to isomorphism as there are comonoid structures on $y^2 + y$.

Solution to Exercise 7.52.

As in Definition 7.49, we have a monoidal category $(\mathcal{C}, y, \triangleleft)$ with comonoids $\mathscr{C} := (\mathfrak{c}, \epsilon, \delta)$ and $\mathscr{C}' := (\mathfrak{c}', \epsilon', \delta')$ and a comonoid morphism $F\colon \mathscr{C} \to \mathscr{C}'$ (really a morphism $F\colon \mathfrak{c} \to \mathfrak{c}'$ in $\mathcal{C}$). Let's throw in another comonoid $\mathscr{C}'' := (\mathfrak{c}'', \epsilon'', \delta'')$ and another comonoid morphism $G\colon \mathscr{C}' \to \mathscr{C}''$ (really a morphism $G\colon \mathfrak{c}' \to \mathfrak{c}''$ in $\mathcal{C}$).

1. To show that the identity morphism $\mathrm{id}_{\mathfrak{c}}\colon \mathfrak{c} \to \mathfrak{c}$ is a comonoid morphism $\mathscr{C} \to \mathscr{C}$, we must check that it preserves erasure by showing that (7.50) commutes, then check that it preserves

duplication by showing that (7.50) commutes:

$$
\begin{tikzcd}
\mathfrak{c} \arrow[r, equal, "\mathrm{id}_{\mathfrak{c}}"] \arrow[d, "\epsilon"'] & \mathfrak{c} \arrow[d, "\epsilon"] \\
y \arrow[r, equal] & y
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathfrak{c} \arrow[r, equal, "\mathrm{id}_{\mathfrak{c}}"] \arrow[d, "\delta"'] & \mathfrak{c} \arrow[d, "\delta"] \\
\mathfrak{c} \triangleleft \mathfrak{c} \arrow[r, equal, "\mathrm{id}_{\mathfrak{c}} \triangleleft \mathrm{id}_{\mathfrak{c}}"] & \mathfrak{c} \triangleleft \mathfrak{c}.
\end{tikzcd}
$$

But they do commute, since $\mathrm{id}_{\mathfrak{c}}$ is the identity on $\mathfrak{c}$ and $\mathrm{id}_{\mathfrak{c}} \triangleleft \mathrm{id}_{\mathfrak{c}}$ is the identity on $\mathfrak{c} \triangleleft \mathfrak{c}$.

2. To show that the composite $F \mathbin{⨟} G \colon \mathfrak{c} \to \mathfrak{c}''$ of the two comonoid morphisms F and G is itself a comonoid morphism $\mathscr{C} \to \mathscr{C}''$, we check that it preserves erasure by showing that (7.50) commutes, then check that it preserves duplication by showing that (7.50) commutes:

$$
\begin{tikzcd}
\mathfrak{c} \arrow[r, "F ⨟ G"] \arrow[d, "\epsilon"'] & \mathfrak{c}'' \arrow[d, "\epsilon''"] \\
y \arrow[r, equal] & y
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathfrak{c} \arrow[r, "F ⨟ G"] \arrow[d, "\delta"'] & \mathfrak{c}'' \arrow[d, "\delta''"] \\
\mathfrak{c} \triangleleft \mathfrak{c} \arrow[r, "(F ⨟ G) \triangleleft (F ⨟ G)"] & \mathfrak{c}'' \triangleleft \mathfrak{c}''.
\end{tikzcd}
$$

But we can rewrite these squares like so, using the fact that $(F ⨟ G) \triangleleft (F ⨟ G) = (F \triangleleft F) ⨟ (G \triangleleft G)$ on the right:

$$
\begin{tikzcd}
\mathfrak{c} \arrow[r, "F"] \arrow[d, "\epsilon"'] & \mathfrak{c} \arrow[r, "G"] \arrow[d, "\epsilon'"'] & \mathfrak{c}'' \arrow[d, "\epsilon''"] \\
y \arrow[r, equal] & y \arrow[r, equal] & y
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathfrak{c} \arrow[r, "F"] \arrow[d, "\delta"'] & \mathfrak{c}' \arrow[r, "G"] \arrow[d, "\delta"'] & \mathfrak{c}'' \arrow[d, "\delta'"] \\
\mathfrak{c} \triangleleft \mathfrak{c} \arrow[r, "F \triangleleft F"] & \mathfrak{c}' \triangleleft \mathfrak{c}' \arrow[r, "G \triangleleft G"] & \mathfrak{c}'' \triangleleft \mathfrak{c}''.
\end{tikzcd}
$$

Then the left square in each diagram commutes because F is a comonoid morphism, while the right square in each diagram commutes because G is a comonoid morphism. So both diagrams commute.

Solution to Exercise 7.60.

Here $F \colon \mathcal{C} \nrightarrow \mathcal{D}$ and $G \colon \mathcal{D} \nrightarrow \mathcal{E}$ are retrofunctors in $\mathbf{Cat}^{\sharp}$.

1. The identity retrofunctor $\mathrm{id}_{\mathcal{D}}$ on $\mathcal{D}$ should correspond to the identity lens on the carrier of $\mathcal{D}$, which is the identity on both positions (objects) and directions (morphisms). So $\mathrm{id}_{\mathcal{D}}$ sends each object $d \in \mathrm{Ob}\,\mathcal{D}$ to itself, while $(\mathrm{id}_{\mathcal{D}})_d^{\sharp} \colon \mathcal{D}[d] \to \mathcal{D}[d]$ sends each morphism out of d to itself as well.
2. The composite retrofunctor $F ⨟ G \colon \mathcal{C} \nrightarrow \mathcal{D}$ should correspond to the composite of F as a lens between the carriers of $\mathcal{C}$ and $\mathcal{D}$ with G as a lens between the carriers of $\mathcal{D}$ and $\mathcal{E}$. So on objects, $F ⨟ G$ sends each $c \in \mathrm{Ob}\,\mathcal{C}$ to $G(Fc) \in \mathrm{Ob}\,\mathcal{E}$. Then given $c \in \mathrm{Ob}\,\mathcal{C}$, the on-morphisms function $(F ⨟ G)_c^{\sharp} \colon \mathcal{E}[G(Fc)] \to \mathcal{C}[c]$ is the composite of on-directions functions

$$\mathcal{E}[G(Fc)] \xrightarrow{G^{\sharp}_{Fc}} \mathcal{D}[Fc] \xrightarrow{F^{\sharp}_{c}} \mathcal{C}[c],$$

sending each morphism h with domain $G(Fc)$ to $F_c^{\sharp}\left(G^{\sharp}_{Fc} h\right)$.

Solution to Exercise 7.62.

We want to show that categories $\mathcal{C}$ and $\mathcal{D}$ are isomorphic in **Cat** if and only if they are isomorphic in $\mathbf{Cat}^{\sharp}$. First, assume that $\mathcal{C}$ and $\mathcal{D}$ are isomorphic in **Cat**, so that there exist mutually inverse functors $F \colon \mathcal{C} \to \mathcal{D}$ and $G \colon \mathcal{D} \to \mathcal{C}$. Then we can define a retrofunctor $H \colon \mathcal{C} \nrightarrow \mathcal{D}$ such that for each $c \in \mathcal{C}$ we have $Hc \coloneqq Fc \in \mathcal{D}$, and for each $g \in \mathcal{D}[Fc]$ we have $H_c^{\sharp} g \coloneqq Gg \in \mathcal{C}[GFc] = \mathcal{C}[c]$. We can verify that H really is a retrofunctor: it preserves identities and composition because G does, and it preserves codomains because $H \operatorname{cod} H_c^{\sharp} g = F \operatorname{cod} Gg = FG \operatorname{cod} g = \operatorname{cod} g$. Analogously, we can define a retrofunctor $K \colon \mathcal{D} \nrightarrow \mathcal{C}$ with $Kd \coloneqq Gd$ and $K_d^{\sharp} f \coloneqq Ff$ for each $d \in \mathcal{D}$ and $f \in \mathcal{C}[Gd]$. Then $H ⨟ K$ is equal to $F ⨟ G$ both on objects and on morphisms, so it is the identity retrofunctor on $\mathcal{C}$; analogously, $K ⨟ H$ is the identity retrofunctor on $\mathcal{D}$. Thus $\mathcal{C}$ and $\mathcal{D}$ are isomorphic in $\mathbf{Cat}^{\sharp}$.

Conversely, assume $\mathcal{C}$ and $\mathcal{D}$ are isomorphic in $\mathbf{Cat}^{\sharp}$, so that there exist mutually inverse retrofunctors $H \colon \mathcal{C} \nrightarrow \mathcal{D}$ and $K \colon \mathcal{D} \nrightarrow \mathcal{C}$. Given objects c, c' and a morphism $f \colon c \to c'$ in $\mathcal{C}$, we have that $KHc = c$, so $K^\sharp_{Hc}$ is a function $\mathcal{C}[c] \to \mathcal{D}[Hc]$. In particular, $K^\sharp_{Hc} f$ is a morphism in $\mathcal{D}$ whose domain is Hc and whose codomain satisfies $K \operatorname{cod} K^\sharp_{Hc} f = \operatorname{cod} f = c'$, and thus $\operatorname{cod} K^\sharp_{Hc} f = Hc'$. Hence we can define a functor $F\colon \mathcal{C} \to \mathcal{D}$ such that for each $c \in \mathcal{C}$ we have $Fc := Hc \in \mathcal{D}$, and for each morphism $f\colon c \to c'$ in $\mathcal{C}$ we have $Ff := K^\sharp_{Hc} f\colon Hc \to Hc'$. Functoriality follows from the fact that K preserves identities and composition. Analogously, we can define a functor $G\colon \mathcal{D} \to \mathcal{C}$ with $Gd := Kd$ for each $d \in \mathcal{D}$ and $Gg := H^\sharp_{Kd} g$ for each $g\colon d \to d'$ in $\mathcal{D}$. Then $F \mathbin{⨾} G$ is equal to $H \mathbin{⨾} K$ on objects; on morphisms, it sends each $f\colon c \to c'$ in $\mathcal{C}$ to $H^\sharp_{KHc}\left(K^\sharp_{Hc} f\right) = H^\sharp_c \left(K^\sharp_{Hc} f\right) = (K \mathbin{⨾} H)^\sharp_c f = f$ itself. So $F \mathbin{⨾} G$ is the identity retrofunctor on $\mathcal{C}$. Analogously, $G \mathbin{⨾} F$ is the identity functor on $\mathcal{D}$. Thus $\mathcal{C}$ and $\mathcal{D}$ are isomorphic in **Cat**.

Solution to Exercise 7.64.

1. We actually already showed that y has a unique comonoid structure, corresponding to the category with 1 object and no nonidentity morphisms (which we will also denote by y), in Exercise 7.39, for the case of $S := 1$.
2. For any category $\mathcal{C}$, there is a unique retrofunctor $\mathcal{C} \nrightarrow y$: it sends every object in $\mathcal{C}$ to the only object in y, and it sends the only morphism in y, an identity morphism, to each identity morphism in $\mathcal{C}$.
3. By Example 7.63, a retrofunctor from a category $\mathcal{C}$ to the discrete category on $S \in \mathbf{Set}$ is a way of assigning each state in $\mathcal{C}$ a label in S. In this case, y is the discrete category on 1, so there is only 1 label to choose from; hence there is always just 1 way to assign the labels.

Solution to Exercise 7.66.

Given a retrofunctor $F\colon \mathcal{C} \nrightarrow \mathcal{A}$, where $\mathcal{A}$ is the walking arrow category as in Example 7.65, assume $Fc = s$ for all $c \in \mathcal{C}$. By Example 7.65, if $Fc = s$ for $c \in \mathcal{C}$, then there must be a morphism from c to an object in $\mathcal{C}$ that F sends to t for $a\colon s \to t$ in $\mathcal{A}$ to be sent back to via $F^\sharp$. But there are no objects in $\mathcal{C}$ that F sends to t. So the only way such a retrofunctor could be defined is if there are no objects in $\mathcal{C}$ that it sends to s, either: we conclude that $\mathcal{C}$ is the empty category.

Solution to Exercise 7.67.

The star-shaped category has $n+1$ objects, one of which is "central" in the sense that it maps uniquely to every object; the other objects have only identity maps.

1. A retrofunctor $\mathsf{C} \nrightarrow \mathsf{y}^{n+1} + \mathsf{n}y$ comprises an assignment of a label to each object in $\mathcal{C}$: either it assigns "center" or it assigns an element of $\{1, 2, \ldots, n\}$. If it assigns "center", the object is equipped with n-many morphisms, with the ith one having as its codomain an object labeled i.
2. A retrofunctor $\mathcal{C} \to (\mathbb{N}, \leq)$ comprises an assignment of a natural number label to each object in $\mathcal{C}$, as well as a choice of morphism in $\mathcal{C}$ from each object labeled n to some object labeled $n+1$.
3. A retrofunctor $\mathcal{C} \to (\mathbb{N}, \leq)$ comprises an assignment of a natural number label to each object in $\mathcal{C}$, as well as a choice of morphism in $\mathcal{C}$ from each object labeled $n+1$ to some object labeled n.

Solution to Exercise 7.69.

1. Every object $c \in \mathcal{CS}$ in the commutative square must be labeled s or t. If c is labeled t, there are no further restrictions, since the only arrow emanating from t is the identity. If c is labeled s, then we must choose an outgoing arrow from c to an object labeled t. So the possible retrofunctors are

$$
\begin{tikzcd} t & t \\ t & t \end{tikzcd}
\quad
\begin{tikzcd} s \arrow[r] & t \\ t & t \end{tikzcd}
\quad
\begin{tikzcd} s \arrow[d] & t \\ t & t \end{tikzcd}
\quad
\begin{tikzcd} s \arrow[dr] & t \\ t & t \end{tikzcd}
\quad
\begin{tikzcd} t & s \arrow[d] \\ t & t \end{tikzcd}
\quad
\begin{tikzcd} t & t \\ s \arrow[r] & t \end{tikzcd}
$$

$$
\begin{tikzcd} s \arrow[d] & s \arrow[d] \\ t & t \end{tikzcd}
\quad
\begin{tikzcd} s \arrow[dr] & s \arrow[d] \\ t & t \end{tikzcd}
\quad
\begin{tikzcd} s \arrow[r] & t \\ s \arrow[r] & t \end{tikzcd}
\quad
\begin{tikzcd} s \arrow[dr] & t \\ s \arrow[r] & t \end{tikzcd}
\quad
\begin{tikzcd} t & s \arrow[d] \\ s \arrow[r] & t \end{tikzcd}
\quad
\begin{tikzcd} s \arrow[dr] & s \arrow[d] \\ s \arrow[r] & t \end{tikzcd}
$$

2. Every object $a \in \mathcal{A}$ in the walking arrow must be labeled w, x, y, or z. If a is labeled z, there are no further restrictions. If it is labeled x or y, then a must have a map to an element labeled z; in particular this implies that a must be the source object $s \in \mathcal{A}$. Finally, a cannot be labeled w because then a would need too many outgoing arrows. So the possible retrofunctors are

$$z \qquad z \qquad x \longrightarrow z \qquad y \longrightarrow z$$

Solution to Exercise 7.70.

1. Let $\mathcal{P}$ be a poset. A retrofunctor $y \nrightarrow \mathcal{P}$ represents an maximal element $p \in \mathcal{P}$. Indeed, if there were some $p \leq p'$ with $p' \neq p$, then the retrofunctor would have to send the map $p \to p'$ to some morphism in y whose codomain is sent to p'; this is impossible. But if p is maximal, then there is no obstruction to sending the unique object of y to p.
2. Let $m = \boxed{\bullet^0 \to \cdots \to \bullet^{m-2} \to \bullet^{m-1}}$. By the same reasoning as above, a map $[m] \to [n]$ must send the final object of $[m]$ to the final object of $[n]$. It can send the penultimate object $m-2$ to either the penultimate object $n-2$ or to the final object $n-1$ in $[n]$. Repeating in this way, we see that there 2^m many retrofunctors. For example, the retrofunctors $[2] \to [2]$ are those labeled by $(2,2,2)$, $(1,2,2)$, $(1,1,2)$, and $(0,1,2)$.

Solution to Exercise 7.73.

We seek the number of arrow fields on 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 $2 \cdot 1 = 2$ arrow fields.

Solution to Exercise 7.74.

1. A retrofunctor $\mathcal{C} \nrightarrow y^{\mathbb{Z}}$ assigns to each object $c \in \mathcal{C}$ and integer $n \in \mathbb{Z}$ an emanating arrow $c.n \in \mathcal{C}$ to some other object, with the property that $c.0 = c$ and $c.n.n' = c.(n+n')$. But this is overkill. Indeed, it is enough to assign the $c.1$ arrow and to check that for every object c' there exists a unique object c with $\operatorname{cod}(c.1) = c'$.
2. We seek a canonical retrofunctor $y^{\mathbb{Z}} \nrightarrow y^{\mathbb{N}}$. The canonical inclusion $i\colon \mathbb{N} \hookrightarrow \mathbb{Z}$ gives rise to a lens ι from $y^{\mathbb{Z}}$ to $y^{\mathbb{N}}$, whose sole on-directions function $\iota^\sharp\colon \mathbb{N} \hookrightarrow \mathbb{Z}$ coincides with i. We verify that ι is a retrofunctor: it preserves identities, as $\iota^\sharp 0 = 0$; it automatically preserves codomains; and it preserves composites, given by addition in either monoid, as $\iota^\sharp(m+n) = m+n = \iota^\sharp(m) + \iota^\sharp(n)$.

Solution to Exercise 7.75.

1. Retrofunctors $y^M \to y^N$ are the same as monoid homomorphisms $N \to M$. See Proposition 7.79.
2. No! This is the weirdest thing about retrofunctors. For example, there is a unique retrofunctor $y \to y^2 + y$ from the walking object to the walking arrow, so if we reverse the arrows, there is no longer a retrofunctor that acts the same on objects.

Solution to Exercise 7.76.

We are given a monoid $(M, e, *)$, a set S, and an M-action $\alpha\colon M \times S \to S$. The category Sy^M has e as the identity on each $s \in S$; the codomain of the map labeled m emanating from s is $\alpha(s, m)$, and composition is given by $*$. The projection $Sy^M \to y^M$ sends the identity back to the identity, trivially preserves codomains, and also preserves composition, so it is a retrofunctor.

Solution to Exercise 7.78.

This works!

Solution to Exercise 7.81.

A continuous arrow field on $\mathcal{C}$ assigns to each object $c \in \mathcal{C}$ and each real number $r \in \mathbb{R}$ a morphism $c.r$ emanating from c. These have the property that $c.0$ is the identity on c and that $(c.r).r' = c.(r+r')$. In other words, you can evolve c forward or backward in time by any $r \in \mathbb{R}$, and this works as expected.

Solution to Exercise 7.86.

Given a bijection of sets $S \cong T \times U$, we seek a unique retrofunctor $Sy^S \nrightarrow Ty^T$ whose on-positions function get is given by the product projection $S \cong T \times U \to T$. Such a retrofunctor should also have an on-directions function put: $S \times T \to S$ such that the three laws from Example 7.85 are satisfied. With $S \cong T \times U$, such a function is uniquely determined by its components

$$S \times T \xrightarrow{\text{put}} S \xrightarrow{\text{get}} T \qquad \text{and} \qquad S \times T \xrightarrow{\text{put}} S \xrightarrow{\pi} U,$$

where $\pi \colon S \cong T \times U \to U$ is the other product projection. The left component is determined by the put-get law, which specifies the behavior of the composite put $⨟$ get: it should send $(s, t) \mapsto t$. Identifying S with $T \times U$, the get-put law for $s = (t, u) \in T \times U$ (so $\text{get}(t, u) = t$) reads as

$$\text{put}((t, u), t) = (t, u),$$

while the put-put law reads as

$$\text{put}(\text{put}((t, u), t'), t'') = \text{put}((t, u), t'')$$

for $t', t'' \in T$. Applying get to both sides, we observe that the first coordinate (in T) of either side of each equation are automatically equal when the put-get law holds. So we are really only concerned with the second coordinates of either side (in U); applying π to both sides yields

$$\pi(\text{put}((t, u), t)) = u \qquad \text{and} \qquad \pi(\text{put}(\text{put}((t, u), t'), t'')) = \pi(\text{put}((t, u), t''))$$

Solution to Exercise 7.87.

1. There are 6 product projections $\mathsf{3} \to \mathsf{3}$, namely the three automorphisms, so the answer is 6.
2. There are 6 product projections $\mathsf{4} \to \mathsf{2}$, so the answer is 6.

Solution to Exercise 7.89.

Yes, it is a retrofunctor. One sees easily that it sends identities back to identities and composites back to composites. The codomain of $f ⨟ g$ is simply $f ⨟ g$ as an element of $\mathfrak{c}[i]$, and it is sent forward to $\text{cod}(f ⨟ g)$, so the map preserves codomains.

Solution to Exercise 7.91.

1. Every object in $\mathcal{U}$ must be labeled with either 1 or 2, but there are no other requirements.
2. If retrofunctors $\mathcal{U} \to \mathcal{C}$ are always in bijection with objects in $\mathcal{C}$, then with $\mathcal{C} = \mathsf{2}y$, we have a bijection $\mathsf{2} \cong \mathsf{2}^{\text{Ob}(\mathcal{U})}$ by part 1, so $\mathcal{U}$ has one object.
3. Take $\mathcal{D}$ to be the walking arrow. Then if $\mathcal{U}$ has only one object, it cannot be sent to the source object of $\mathcal{D}$ because the emanating morphism would have nowhere to go. Hence the unique object of $\mathcal{U}$ must be sent to the target object of $\mathcal{D}$. But there is only one such retrofunctor, whereas there are two objects in $\mathcal{D}$. We conclude that objects are not reprsentable in $\mathbf{Cat}^\sharp$ the way they are in **Cat**.
4. No, there is no such $\mathcal{V}$. There is exactly one retrofunctor $0 \nrightarrow \mathcal{V}$, but there are no objects of 0.

Solution to Exercise 7.93.

There are two: one of which sends $s \mapsto u$ and $t \mapsto v$ and the other of which sends both $s, t \mapsto v$.

Solution to Exercise 7.94.

1. Take the category $\mathcal{C}'$ that has the same objects as $\mathcal{C}$ and almost the same arrows, except that it has one more arrow i_c from each object $c \in \mathcal{C}'$ to itself. This new arrow is the identity. The composites of all the old arrows in $\mathcal{C}'$ are exactly as they are in $\mathcal{C}$. The old identity id_c^{old} is no longer an identity because $\text{id}_c^{\text{old}} ⨟ i = \text{id}_c^{\text{old}} \neq i$.
2. Given a retrofunctor $\varphi \colon \mathfrak{c} \nrightarrow \mathfrak{d}$, we get a retrofunctor $\mathfrak{c}y \nrightarrow \mathfrak{d}y$ that acts the same on objects and all the old arrows and sends the new identities back to the new identities. It preserves identities and composites going backward and codomains going forward, so it's a retrofunctor.

3. It is a monad: there is an obvious unit map $\mathfrak{c} \to \mathfrak{c}y$ and a multiplication map $\mathfrak{c}yy \to \mathfrak{c}y$ that sends the new identity back to the newest identity.

Solution to Exercise 7.95.

1. Counterexample: take $\mathfrak{c} = 0$. Then the unique lens $f\colon \mathfrak{c} \to \mathfrak{d}$ is a retrofunctor and so is $f \fatsemi g$ for any lens $g\colon \mathfrak{d} \to \mathfrak{e}$, but some lenses are not retrofunctors.
2. Counterexample: take $\mathfrak{e} = y$. Then there is a unique retrofunctor $\epsilon\colon \mathfrak{c} \to y$. As long as f is copointed, meaning it sends identities backward to identities, then $f \fatsemi g$ will be a retrofunctor. But not every copointed lens is a retrofunctor.

Solution to Exercise 7.101.

In general, a functor $F\colon \mathcal{C} \to \mathcal{D}$ is a discrete opfibration iff, for every object $c \in \mathcal{C}$, the induced map $F[c]\colon \mathcal{C}[c] \to \mathcal{D}[Fc]$ is a bijection.

In our case, if π and π' are discrete opfibrations then for any $s \in \mathcal{S}$ we have that both the second map and the composite in $\mathcal{S}[s] \xrightarrow{F[s]} \mathcal{S}'[Fs] \xrightarrow{\pi'[Fs]} \mathcal{C}[\pi s]$ are bijections, so the first map is too.

Solution to Exercise 7.102.

1. The identity map id_i satisfies $\pi(\mathrm{id}_i) = \mathrm{id}_{\pi(i)}$, so $\overline{\mathrm{id}_{\pi(i)}} = \mathrm{id}_i$ by the uniqueness-of-lift condition in Definition 7.99.
2. The morphism $\overline{f} \fatsemi \overline{g}$ satisfies $\pi(\overline{f} \fatsemi \overline{g}) = \pi(\overline{f}) \fatsemi \pi(\overline{g}) = f \fatsemi g$, so again this follows by uniqueness of lift.
3. To see that π is a retrofunctor, we need to understand and verify conditions about its action forward on objects and backward on morphisms. Its action forward on objects is that of π as a functor. Given an object $s \in \mathcal{S}$, the action of π backward on morphisms is the lift operation $\pi^\sharp_s(f) \coloneqq \overline{f}$. This preserves identity by part 1, composition by part 2, and codomains because $\pi(\mathrm{cod}(\pi^\sharp_s(f))) = \mathrm{cod}(\pi(\pi^\sharp_s(f))) = \mathrm{cod}(f)$.

Solution to Exercise 7.105.

1. The functor action on objects and morphisms is defined in the exercise statement, so it suffices to check that it preserves identities and composites. But this too is obvious: $\pi(\mathrm{id}) = \mathrm{id}$ and $\pi(f \fatsemi g)$ is $f \fatsemi g$.
2. Given an object $(c, x) \in \int^{\mathcal{C}} I$ and $f\colon c \to c'$, we need to check that there is a unique object $s \in \int^{\mathcal{C}} I$ and morphism $\overline{f}\colon (c, x) \to s$ with $\pi(\overline{f}) = f$. Using $x' \coloneqq I(f)(x)$ and $s \coloneqq (c', x')$ and $\overline{f} \coloneqq f$, we do indeed get $\pi(\overline{f}) = f$ and we find that it is the only possible choice.

Solution to Exercise 7.106.

1. Given a natural transformation $\alpha\colon I \to J$, we need a functor $\int^{\mathcal{C}} \alpha\colon \int^{\mathcal{C}} I \to \int^{\mathcal{C}} J$. On objects have it send $(c, x) \mapsto (c, \alpha_c x)$, where $x \in I(c)$ so $\alpha_c x \in J(c)$; on morphisms have it send $f \mapsto f$, a mapping which clearly preserves identities and composition.
2. To see that $\int^{\mathcal{C}} \alpha$ is a morphism between discrete opfibrations, one only needs to check that it commutes with the projections to $\mathcal{C}$, but this is obvious: the mapping $(c, x) \mapsto (c, \alpha_c x)$ and the mapping $f \mapsto f$ preserve the objects and morphisms of $\mathcal{C}$.
3. It is clear that $\int^{\mathcal{C}}$ preserves identities and composition in $\mathbf{Set}^{\mathcal{C}}$, so we have verified that this is functor.

Solution to Exercise 7.107.

Given a graph $G \coloneqq (\mathrm{src}, \mathrm{tgt}\colon A \rightrightarrows V)$ and functor $S\colon \mathcal{G} \to \mathbf{Set}$, define a new graph H as the following diagram of sets:

$$\sum_{a \in A} S(\mathrm{src}(a)) \overset{(\mathrm{src},\mathrm{id})}{\underset{(\mathrm{tgt},S(a))}{\rightrightarrows}} \sum_{v \in V} S(v)$$

In other words, it is a graph for which a vertex is a pair (v, s) with $v \in V$ a G-vertex and $s \in S(v)$, and for which an arrow is a pair (a, s) with $a \in A$ a G-arrow, say $a\colon v \to v'$, and $s \in S(v)$ is an element over the source of that arrow. With this notation, the arrow (a, s) has as source vertex (v, s) and has as target vertex $(v', S(a)(s))$ where $S(a)\colon S(v) \to S(v')$ is the function given by the functor $S\colon \mathcal{G} \to \mathbf{Set}$.

It remains to see that the free category on H is isomorphic to $\int^{\mathcal{G}} S$. We first note that it has the same set of objects, namely $\sum_{v \in V} S(v)$. A morphism in $\int^{\mathcal{G}} S$ can be identified with an object (v, s) together with a morphism $f\colon v \to v'$ in $\mathcal{G}$, but this is just a length-n sequence $(a_1, \ldots, a_n)$ of arrows, with $v = \mathrm{src}(a_1)$, $\mathrm{tgt}(a_i) = \mathrm{src}(a_{i+1})$ for $1 \leq i < n$, and $\mathrm{tgt}(a_n) = v'$. This is the same data as a morphism in the free category on H.

Chapter 8

Categorical properties of polynomial comonoids

While we defined the category $\mathbf{Cat}^\sharp$ of categories and retrofunctors in the last chapter, we have only begun to scratch the surface of the properties it satisfies. As the category of comonoids in **Poly**, equipped with a canonical forgetful functor $U\colon \mathbf{Cat}^\sharp \to \mathbf{Poly}$ sending each polynomial to its carrier, $\mathbf{Cat}^\sharp$ inherits many of the categorical properties satisfied by **Poly**. Underpinning this inheritance is the fact that U has a right adjoint, a cofree functor $\mathbf{Poly} \to \mathbf{Cat}^\sharp$. We will introduce this adjunction in the first section of this chapter. Then we will discuss how many of the categorical properties of **Poly**, including much of what we covered in Chapter 5, play nicely with restriction to $\mathbf{Cat}^\sharp$. Finally, we will touch on other constructions we can make over the comonoids in **Poly**, such as their comodules and coalgebras.

8.1 Cofree comonoids

Consider a dynamical system $\varphi\colon \mathfrak{s} \to p$ with state system $\mathfrak{s}$ and interface p. In Section 7.1.5, we posed the question of whether there was a single morphism that could capture all the information encoded by the family of lenses $\mathrm{Run}_n(\varphi)\colon \mathfrak{s} \to p^{\triangleleft n}$ for all $n \in \mathbb{N}$, defined as the composite

$$\mathfrak{s} \xrightarrow{\delta^{(n)}} \mathfrak{s}^{\triangleleft n} \xrightarrow{\varphi^{\triangleleft n}} p^{\triangleleft n}$$

that models n runs through the system φ.

It turns out that there is: the key is that there is a natural way to interpret every polynomial p as a category $\mathscr{T}_p$, so that retrofunctors into $\mathscr{T}_p$ are exactly lenses into p. In other words, $\mathscr{T}_p$ will turn out to be the *cofree comonoid* (or *cofree category*) *on* p. 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.

We'll go through the construction of the cofree comonoid and its implications in this section, featuring a purely formal proof of the fact the forgetful functor $U\colon \mathbf{Cat}^\sharp \to \mathbf{Poly}$ has a right adjoint $\mathscr{T}_-\colon \mathbf{Poly} \to \mathbf{Cat}^\sharp$, where for each $p \in \mathbf{Poly}$, the carrier $\mathfrak{t}_p := U\mathscr{T}_p$ of the category $\mathscr{T}_p$ is given by the limit of the following diagram:

$$
\begin{tikzcd}
y \arrow[d] & p \arrow[d] & p^{\triangleleft 2} \arrow[d] & p^{\triangleleft 3} \arrow[d] & \cdots \\
1 & p \triangleleft 1 \arrow[l, "!"'] & p^{\triangleleft 2} \triangleleft 1 \arrow[l, "p \triangleleft !"'] & p^{\triangleleft 3} \triangleleft 1 \arrow[l, "p^{\triangleleft 2} \triangleleft !"'] & \cdots . \arrow[l]
\end{tikzcd}
\tag{8.1}
$$

Thus, we will show in Theorem 8.45 that U and $\mathscr{T}_-$ form a forgetful-cofree adjunction, making $\mathscr{T}_p$ the cofree comonoid on p. But first, let us concretely characterize the canonical comonoid structure on the limit of (8.1), before showing that it is indeed cofree.

8.1.1 The carrier of the cofree comonoid

Let $\mathfrak{t}_p$ be the limit of the diagram (8.1) in **Poly**; it will turn out to be the carrier of the cofree comonoid on p (where p will be an arbitrary polynomial throughout). We could compute this limit directly, but we will be able to describe it more concretely in terms of what we call *trees on* p or p*-trees*: trees comprised of p-corollas. In doing so, we will formalize the tree pictures we have been using to describe polynomials all along.

Trees on polynomials

Definition 8.2 (Tree on a polynomial)**.** Let $p \in \mathbf{Poly}$ be a polynomial. A *tree on* p, or a p*-tree*, is a rooted tree whose every vertex v is assigned a p-position i and a bijection from the children of v to $p[i]$. We denote the set of p-trees by tree_p.

We can think of a p-tree as being "built" out of p-corollas according to these instructions:

To choose a p-tree in tree_p:

1. choose a p-corolla:
   - its root $i_0 \in p(1)$ will be the tree's root, and
   - its leaves in $p[i_0]$ will be the edges out of the root;
2. for each $p[i_0]$-leaf a_1:
   2.1. choose a p-corolla:
      - its root $i_1 \in p(1)$ will be the vertex adjoined to a_1, and
      - its leaves in $p[i_1]$ will be the edges out of that vertex;

   2.2. for each $p[i_1]$-leaf a_2:
      2.2.1. choose a p-corolla:

- its root $i_2 \in p(1)$ will be the vertex adjoined to a_2, and
- its leaves in $p[i_2]$ will be the edges out of that vertex;

2.2.2. for each $p[i_1]$-leaf a_2:

...

Of course, there may eventually be multiple copies of any one p-root as a vertex or p-leaf as an edge in our p-tree, and these vertices and edges are not literally the same. So we should really think of the positions and directions of p involved each step as *labels* for the vertices and edges of a p-tree—although crucially, each $p[i]$-direction must be used as a label exactly once among the edges emanating from a given vertex labeled with i.

Although these instructions continue forever, we could abbreviate them by writing them recursively:

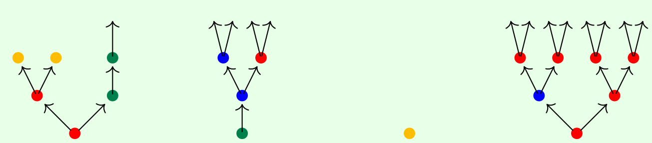
> To choose a p-tree in tree_p:
>
> 1. choose a p-corolla:
>   - its root $i_0 \in p(1)$ will be (the label of) the tree's root, and
>   - its leaves in $p[i_0]$ will be (the labels of) the edges out of the root;
> 2. for each $p[i_0]$-leaf a_1:
>
>   2.1. choose a p-tree in tree_p:
>   - it will be the subtree whose root is adjoined to a_1.

We would like to draw some examples of p-trees, but note that a p-tree can have infinite height—in fact, it always will unless every one of its branches terminates at a p-corolla with no leaves, i.e. a position with an empty direction set. This means that plenty of p-trees cannot be drawn, even when all the position- and direction-sets of p are finite; but we can instead consider finite-height portions of them that we will call *pretrees*.

Pretrees on polynomials

Before we define pretrees, let's give some examples.

Example 8.3 (A few example p-pretrees). Let $p := \{\bullet, \bullet\}y^2 + \{\bullet\}y + \{\bullet\}$. Here are four partially constructed p-trees:



(8.4)

Here only the third one—the single yellow dot—would count as an element of tree_p. After all, in Definition 8.2, when we speak of a tree on p, we mean a tree for which every vertex is a position in p with all of its emanating directions filled by another position in p. Since three of the four trees shown in (8.4) have leaves emanating from the top that have not been filled by any p-corollas, these trees are not elements of tree_p.

However, each of these trees can be extended to an actual element of tree_p by continually filling in each open leaf with another p-corolla. These might continue forever—or, if you're lazy, you could just cap them all off with the direction-less yellow dot.

Exercise 8.5 (Solution here). Let $q \coloneqq y^2 + 3y^1$. Are there any finite q-trees? If not, could there be any vertices of a given q-tree with finitely many descendents? $\Diamond$

The trees in (8.4) can all be obtained by following just the first 3 levels of instructions for building a p-tree (in fact, exactly as many instructions as we initially wrote out). On the other hand, we know from Section 6.1.3 that such a tree represents a position of $p^{\triangleleft 3}$, whose directions are its height-3 leaves—and this is true for any $n \in \mathbb{N}$ in place of 3. So these trees are still important to our theory; but since they are not always complete p-trees, we will call them something else.

Definition 8.6. Given $p \in \mathbf{Poly}$, a *stage-n pretree* (or *$p^{\triangleleft n}$-pretree*) is defined to be an element of $p^{\triangleleft n}(1)$. For each $i \in p^{\triangleleft n}(1)$ the *height-n leaves* of i is defined to be the set $p^{\triangleleft n}[i]$.

Remark 8.7. In Definition 8.6, we use *stage* instead of *height* to allow for the fact that a $p^{\triangleleft n}$-pretree may not reach its maximum height n if all of its branches terminate early. For example, the yellow dot in (8.4) is a $p^{\triangleleft 1}$-pretree, but it is also a $p^{\triangleleft 2}$-pretree, a $p^{\triangleleft 50}$-pretree, and indeed a p-tree.

Note that for any polynomial $p \in \mathbf{Poly}$ there is exactly one stage-0 pretree on p for any $p \in \mathbf{Poly}$, because $p^{\triangleleft 0}(1) = y(1) = 1$.

Example 8.8 (Trimming pretrees). Since $p^{\triangleleft 1}(1) \cong p(1)$ and $p^{\triangleleft 0}(1) \cong y(1) \cong 1$, the unique function $!\colon p(1) \to 1$ can be thought of as a function from $p^{\triangleleft 1}$-positions to $p^{\triangleleft 0}$-positions, or equivalently a function from stage-1 pretrees (i.e. corollas) to stage-0 pretrees on p. We can interpret this function as taking a corolla and "stripping away" its leaves along with the position-label on its root, leaving only a single unlabeled root: a stage-0 pretree.

This deceptively simple function has a surprising amount of utility when combined with other maps. For any $n \in \mathbb{N}$, we can take the composition product in **Poly** of the identity on $p^{\triangleleft n}$ and !, interpreted as a lens between constant polynomials, to obtain a lens $p^{\triangleleft n} \triangleleft !\colon p^{\triangleleft n} \triangleleft p(1) \to p^{\triangleleft n} \triangleleft 1$, or equivalently a function $p^{\triangleleft n}(!)\colon p^{\triangleleft n+1}(1) \to p^{\triangleleft n}(1)$ from stage-$(n+1)$ pretrees to stage-n pretrees on p.

We can deduce the behavior of this function on stage-$(n+1)$ pretrees from what we know about how the composition product interacts with pretrees on p. The identity lens on $p^{\triangleleft n}$ keeps the lower n levels of each $p^{\triangleleft(n+1)}$-pretree intact, while ! will "strip away" the $p^{\triangleleft(n+1)}$-pretree's height-$(n+1)$ leaves, along with all the position-labels on its height-n vertices. Thus $p^{\triangleleft n}(!)$ is the function sending every stage-$(n+1)$ pretree on p to its stage-n pretree, effectively *trimming it down* a level.

We can go even further: composing several such functions yields a composite function

$$p^{\triangleleft n}(1) \xleftarrow{p^{\triangleleft n}(!)} p^{\triangleleft(n+1)}(1) \leftarrow \cdots \leftarrow p^{\triangleleft(n+k-1)}(1) \xleftarrow{p^{\triangleleft(n+k-1)}(!)} p^{\triangleleft(n+k)}(1) \tag{8.9}$$

sending every stage-$(n+k)$ pretree on p to its stage-n pretree by trimming off its top k levels. We will see these functions again shortly.

Trees as a limit of pretrees

Before we go any further in the theory of p-trees, let us look at some more examples.

Example 8.10 (A few more actual p-trees). Keeping $p := \{\bullet, \bullet\}y^2 + \{\bullet\}y + \{\bullet\}$, here are some elements of tree_p that we could imagine (or even draw, at least in part):

- The binary tree that's "all red all the time."
- The binary tree where odd layers are red and even layers are blue.
- The binary tree whose root is red, but after which every left child is red and every right child is blue.[1]
- The tree where all the nodes are red, except for the rightmost branch, which (apart from the red root) is always green.
- Any finite tree, where every branch terminates in a yellow dot.
- A completely random tree: 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.

In fact, there are uncountably many trees in tree_p (even just tree_{2y} has cardinality $2^{\mathbb{N}}$), but only countably many can be uniquely characterized in a finite language like English (and of course only finitely many can be uniquely characterized in the time we have!). Thus most elements of tree_p cannot even be described.

Exercise 8.11 (Solution here). For each of the following polynomials p, characterize the set of trees tree_p.

5. $p := 1$.
6. $p := 2$.
7. $p := y$.

[1] To formalize the notions of "left" and "right," we could think of the direction-sets of the red and blue dots as $2 \cong \{\texttt{left}, \texttt{right}\}$, so that out of every vertex there is an edge labeled `left` and an edge labeled `right`.

8. $p \coloneqq y^2$.
9. $p \coloneqq 2y$.
10. $p \coloneqq y + 1$.
11. $p \coloneqq By^A$ for some sets $A, B \in \mathbf{Set}$. ◊

Exercise 8.12 (Solution here).

1. Say we were interested in n-*ary trees*: infinite (unless $n = 0$) rooted trees in which every vertex has n children. Is there a polynomial p for which tree_p is the set of n-ary trees?
2. Now say we wanted to assign each vertex of an n-ary tree a label from a set L. Is there a polynomial q for which tree_q is the set of L-labeled n-ary trees? ◊

From here, a natural question to ask is how the set tree_p of p-trees is related to the set of $p^{\triangleleft n}$-pretrees $p^{\triangleleft n}(1)$ for each $n \in \mathbb{N}$. A look back at Definition 8.6 gives a clue: every p-tree has a stage-n pretree obtained by removing all vertices of height greater than n, yielding a function $\mathsf{tree}_p \to p^{\triangleleft n}(1)$ that we will denote by $\pi^{(n)}$.

Moreover, since the stage-n pretree of a given p-tree agrees with the stage-n pretree of any stage-$(n+k)$ pretree of the p-tree, the functions $\pi^{(n)}$ should commute with our tree-trimming functions $p^{\triangleleft(n+k)}(1) \to p^{\triangleleft n}(1)$ from (8.9) in Example 8.8. In particular, the following diagram commutes for all $n \in \mathbb{N}$:

$$\begin{tikzcd}
\mathsf{tree}_p \arrow[d, "\pi^{(n)}"'] \arrow[dr, "\pi^{(n+1)}"] & \\
p^{\triangleleft n}(1) & p^{\triangleleft(n+1)}(1). \arrow[l, "p^{\triangleleft n}(!)"]
\end{tikzcd}$$

All this says is that if we trim a p-tree down until only $n+1$ levels are left via $\pi^{(n+1)}$, then trimmed off one more level via $p^{\triangleleft n}(!)$, it would be the same as if we had trimmed it down to n levels from the start via $\pi^{(n)}$. This is summarized by the following larger commutative diagram, which contains every function of the form Example 8.8:

$$\begin{tikzcd}
\mathsf{tree}_p \arrow[d, "\pi^{(0)}"'] \arrow[dr, "\pi^{(1)}"] \arrow[drr, "\pi^{(2)}"] \arrow[drrr, "\pi^{(3)}"] & & & \cdots & \\
1 & p(1) \arrow[l, "!"] & p^{\triangleleft 2}(1) \arrow[l, "p(!)"] & p^{\triangleleft 3}(1) \arrow[l, "p^{\triangleleft 2}(!)"] & \cdots . \arrow[l]
\end{tikzcd} \tag{8.13}$$

So tree_p with the functions $\pi^{(n)}$ forms a cone over the the bottom row—in fact, it is the universal cone. Intuitively, this is because a p-tree carries exactly the information that a compatible sequence of $p^{\triangleleft n}$-pretrees does: no more, no less. But we can prove it formally as well.

Exercise 8.14 (Solution here). Prove that tree_p with the functions $\pi^{(n)} \colon \mathsf{tree}_p \to p^{\triangleleft n}(1)$ is the limit of the diagram

$$1 \xleftarrow{\;!\;} p(1) \xleftarrow{p(!)} p^{\triangleleft 2}(1) \xleftarrow{p^{\triangleleft 2}(!)} p^{\triangleleft 3}(1) \longleftarrow \cdots. \tag{8.15}$$

You may use the fact that (8.13) commutes. ◊

If you know about coalgebras for functors, as mentioned in Example 6.67, then you might know the limit $\mathsf{t}_p(1)$ of (8.15) by a different name: it is the *terminal coalgebra for the functor p*, or the *terminal p-coalgebra*, because it is terminal in the category of p-coalgebras, as the following exercise shows.

Exercise 8.16 (Solution here). A *p-coalgebra morphism* between p-coalgebras $\varphi \colon S \to p(S)$ and $\psi \colon T \to p(T)$ (as in Example 6.67) is a function $f \colon S \to T$ such that the square

$$\begin{tikzcd} S \arrow[r, "\varphi"] \arrow[d, "f"'] & p(S) \arrow[d, "p(f)"] \\ T \arrow[r, "\psi"] & p(T) \end{tikzcd}$$

commutes.

1. Choose what you think is a good function $\mathsf{tree}_p \to p(\mathsf{tree}_p)$.
2. Show that tree_p equipped with your function $\mathsf{tree}_p \to p(\mathsf{tree}_p)$ is the terminal object in the category of p-coalgebras and the morphisms between them.
3. Show that the function $\mathsf{tree}_p \to p(\mathsf{tree}_p)$ you chose is a bijection.

◊

Positions of the cofree comonoid

Now recall from Example 5.35 that in **Poly**, the positions of a limit are the limit of the positions. Moreover, in (8.1), every vertical lens $p^{\triangleleft n} \to p^{\triangleleft n} \triangleleft 1$ is, in fact, a *vertical* lens in the sense of Definition 5.51: an isomorphism on positions. So (8.1) on positions collapses down to its bottom row, viewed as a diagram in **Set**. Yet this is precisely the diagram from (8.15), whose limit is tree_p. So tree_p is the position-set of the limit t_p of (8.1): we write $\mathsf{t}_p(1) \cong \mathsf{tree}_p$.

So, as we said above, $\mathsf{t}_p(1)$ is the set carrying the terminal coalgebra of p, but we prefer to think of it as the set of p-trees, for it gives us a concrete way to realize the limit and its projections, as well as a natural interpretation of the directions of t_p at each position.

The directions of the cofree comonoid

Given that $\mathfrak{t}_p$ is the limit of (8.1), Example 5.35 also tells us how to compute its directions: the directions of a limit are the colimit of the directions. But every polynomial in the bottom row of (8.1) has an empty direction-set, and there are no arrows between polynomials in the top row. So the directions of $\mathfrak{t}_p$ are given by a coproduct of the directions of each $p^{\triangleleft n}$.

More precisely, given a p-tree $T \in \mathsf{tree}_p$ whose stage-n pretree for $n \in \mathbb{N}$ is $\pi^{(n)}T$, the direction-set $\mathfrak{t}_p[t]$ is given by the following coproduct:

$$\mathfrak{t}_p[T] \coloneqq \sum_{n \in \mathbb{N}} p^{\triangleleft n}[\pi^{(n)}T].$$

But by Definition 8.6, each $p^{\triangleleft n}[\pi^{(n)}T]$ is the set of height-n leaves of the $p^{\triangleleft n}$-pretree $\pi^{(n)}T$, which in turn is the stage-n pretree of T. So its height-n leaves coincide with the height-n vertices of T. Therefore, we can identify $p^{\triangleleft n}[\pi^{(n)}T]$ with the set of height-n vertices of T; we denote this set by $\mathsf{vtx}_n(T)$.

Since the coproduct above ranges over all $n \in \mathbb{N}$, it follows that $\mathfrak{t}_p[T]$ is the set of *all* vertices in T; we denote this set by $\mathsf{vtx}(T)$. The on-directions function at T of each projection $\mathfrak{t}_p \to p^{\triangleleft n}$ from the limit must then be the canonical inclusion

$$p^{\triangleleft n}[\pi^{(n)}T] \cong \mathsf{vtx}_n(T) \hookrightarrow \mathsf{vtx}(T) \cong \mathfrak{t}_p[T],$$

sending height-n leaves of $\pi^{(n)}T$ to height-n vertices of T.

Here is alternative way to think about the directions of $\mathfrak{t}_p$ and each $p^{\triangleleft n}$ that will be helpful. A defining feature of a rooted tree is that its vertices are in bijection with its finite rooted paths: each vertex gives rise to a unique path to that vertex from the root, and every finite rooted path arises this way. So the directions of $p^{\triangleleft n}$ at a given $p^{\triangleleft n}$-pretree correspond in turn to the rooted paths of that pretree leading to its height-n leaves; indeed, each such direction is comprised of a sequence of n directions of p, which together specify a length-n rooted path up the pretree. Then for $T \in \mathsf{tree}_p$, the direction-set $\mathfrak{t}_p[T]$ consists of every finite rooted path of T.

Since only finite rooted paths correspond to vertices, all our paths will be assumed to be finite from here on out. This is our preferred way to think about directions in $\mathfrak{t}_p$. When we wish to refer to what one might call an infinite (rooted) "path," we will instead call it a (rooted) *ray*.

Exercise 8.17 (Solution here). For each of the following polynomials p, characterize the polynomial $\mathfrak{t}_p$. You may choose to think of the directions of $\mathfrak{t}_p$ either as vertices or as rooted paths. (Note that these are the same polynomials from Exercise 8.11.)

1. $p \coloneqq 1$.
2. $p \coloneqq 2$.
3. $p \coloneqq y$.
4. $p \coloneqq y^2$.

5. $p \coloneqq 2y$.
6. $p \coloneqq y + 1$.
7. $p \coloneqq By^A$ for some sets $A, B \in$ **Set**. $\Diamond$

We summarize the results of this section in the following proposition, thus concretely characterizing the carrier $\mathfrak{t}_p$ of the cofree comonoid on p in terms of p-trees. For reasons that will become clear shortly, we will denote each projection from the limit of (8.1) by $\epsilon_p^{(n)} \colon \mathfrak{t}_p \to p^{\triangleleft n}$. We denote ϵ simply by ϵ.

Proposition 8.18. For $p \in$ **Poly**, let

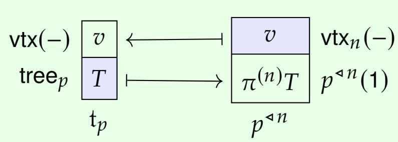
$$\mathfrak{t}_p \coloneqq \sum_{T \in \mathsf{tree}_p} y^{\mathsf{vtx}(T)}$$

be the polynomial whose positions are p-trees and whose directions at each p-tree are the rooted paths. Then $\mathfrak{t}_p$ is the limit of the diagram (8.1), with projections $\epsilon_p^{(n)} \colon \mathfrak{t}_p \to p^{\triangleleft n}$ for every $n \in \mathbb{N}$ making the following diagram commute:

$$\begin{tikzcd}
\mathfrak{t}_p \arrow[d, "\epsilon_p"'] \arrow[dr, "\epsilon_p^{(1)}"'] \arrow[drr, "\epsilon_p^{(2)}"'] \arrow[drrr, "\epsilon_p^{(3)}"'] & & & & \cdots \\
y \arrow[d] & p \arrow[d] & p^{\triangleleft 2} \arrow[d] & p^{\triangleleft 3} \arrow[d] & \cdots \\
1 & p(1) \arrow[l, "!"] & p^{\triangleleft 2}(1) \arrow[l, "p(!)"] & p^{\triangleleft 3}(1) \arrow[l, "p^{\triangleleft 2}(!)"] & \cdots \arrow[l] .
\end{tikzcd} \tag{8.19}$$

The lens $\epsilon_p^{(n)} \colon \mathfrak{t}_p \to p^{\triangleleft n}$ sends each p-tree $T \in \mathsf{tree}_p$ to its stage-n pretree $\pi^{(n)}T$ on positions and each height-n leaf of $\pi^{(n)}T$ to the corresponding height-n rooted path of T on directions.

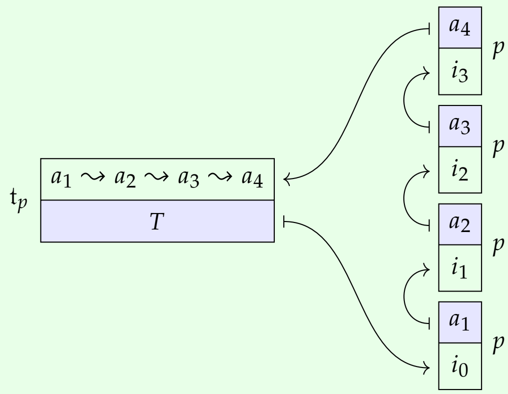
Example 8.20 (Drawing $\epsilon_p^{(n)}$ in polyboxes)*.* Here is $\epsilon_p^{(n)} \colon \mathfrak{t}_p \to p^{\triangleleft n}$ drawn in polyboxes, where we continue to denote the stage-n pretree of a p-tree by $\pi^{(n)}T$:



(8.21)

On the right hand side, v is a height-n leaf of $\pi^{(n)}T$; on the left, v is identified with the corresponding height-n vertex of T.

This isn't the only way we can write this lens in polyboxes, however; polyboxes have special notation for lenses to composites, allowing us to write, say, the $n = 4$ case like so:

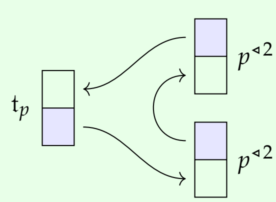


(8.22)

Here we are unpacking the construction of the first 4 levels of T, according to our nested instructions for building p-trees. The p-position i_0 is the label on the root of T, while the $p[i_0]$-direction a_1 specifies one of the edges coming out of it—leading to a height-1 vertex of T labeled i_1, and so on.

The contents of the position-boxes and the arrows going up on the codomain side carry all the data of the bottom 4 levels of T: namely the label i_0 of the root, the label i_1 of the vertex at the end of every direction a_1 out of the root, and so on until i_3. All this specifies a unique $p^{\triangleleft 4}$-position, a $p^{\triangleleft 4}$-pretree, which is the same as the $p^{\triangleleft 4}$-pretree T. Indeed, we can think of $\pi^{(4)}T$ as a shorthand for the gadget comprised of the 4 polyboxes on the right hand side of (8.22) when their blue boxes are yet to be filled. So the position and direction depicted on the codomain side of (8.22) is equivalent to the position and direction depicted on the codomain side of (8.21). This generalizes to all values of n. The direction a_4 emanating from i_3, together with all the data below it, corresponds to the rooted path we've denoted $a_1 \rightsquigarrow a_2 \rightsquigarrow a_3 \rightsquigarrow a_4$ in T.

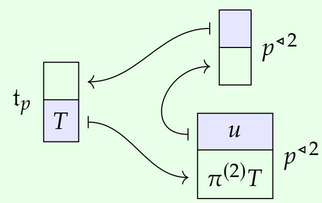
There are even more ways to express $\epsilon_p^{(4)} \colon \mathrm{t}_p \to p^{\triangleleft 4}$ in polyboxes, however. After all, $p^{\triangleleft 4} \cong p^{\triangleleft 2} \triangleleft p^{\triangleleft 2}$. So we ought to be able to draw $\epsilon_p^{(4)}$ as follows:



(8.23)

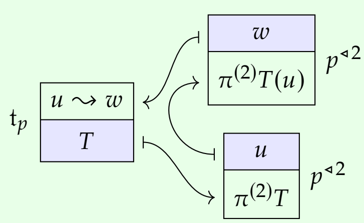
Let's think about how we should fill these boxes. We can still put a p-tree T in the lower left position box. From the commutativity of (8.19), we know the on-positions function $\pi^{(4)}$ of $\epsilon_p^{(4)}$ factors through the on-positions function $\pi^{(2)}$ of $\epsilon_p^{(2)}$, which tells us that $p^{\triangleleft 2}$-pretree that should go in the lower right position box should be $\pi^{(2)}T$: the $p^{\triangleleft 2}$-pretree of T.

Another way to think about this is that the polyboxes for the lower $p^{\triangleleft 2}$ on the right side of (8.23) are equivalent to the polyboxes for the 2 lower copies of p on the right side of (8.22)—just like how the polyboxes for $p^{\triangleleft n}$ in (8.21) are equivalent to the polyboxes for all $n = 4$ copies of p in (8.22). There, the position could be represented with a single $p^{\triangleleft n}$-pretree $\pi^{(n)}T$, and the direction $(a_1, \ldots, a_n)$ is one of its length-n rooted paths. So here, too, we can package the 2 lower copies of p into a single pair of polyboxes for $p^{\triangleleft 2}$, if we let u be the height-2 vertex of T at the end of the rooted path (a_1, a_2):



(8.24)

But then the polyboxes for the 2 upper copies of p from (8.22) should also collapse down to a single pair of polyboxes for $p^{\triangleleft 2}$, with a $p^{\triangleleft 2}$-pretree as its position and a height-2 leaf of that pretree as its direction. Indeed, once we have followed the directions (a_1, a_2) up to the height-2 vertex u, the subtree of T rooted at u is itself a p-tree: it has a label i_2 on its root, out of which we can follow the direction a_3 to reach a height-1 vertex labeled i_3, then follow the direction a_4 to reach a height-2 vertex that we call w. Of course, the vertex labeled i_3 is actually a height-3 vertex of the whole tree T; likewise w corresponds to a height-4 vertex of T. However, from the perspective of the upper copy of $p^{\triangleleft 2}$ in (8.24), we are starting over from u and moving up the subtree of T rooted at u—or, more precisely, the $p^{\triangleleft 2}$-pretree of the subtree rooted at u. So the polyboxes end up looking like this:

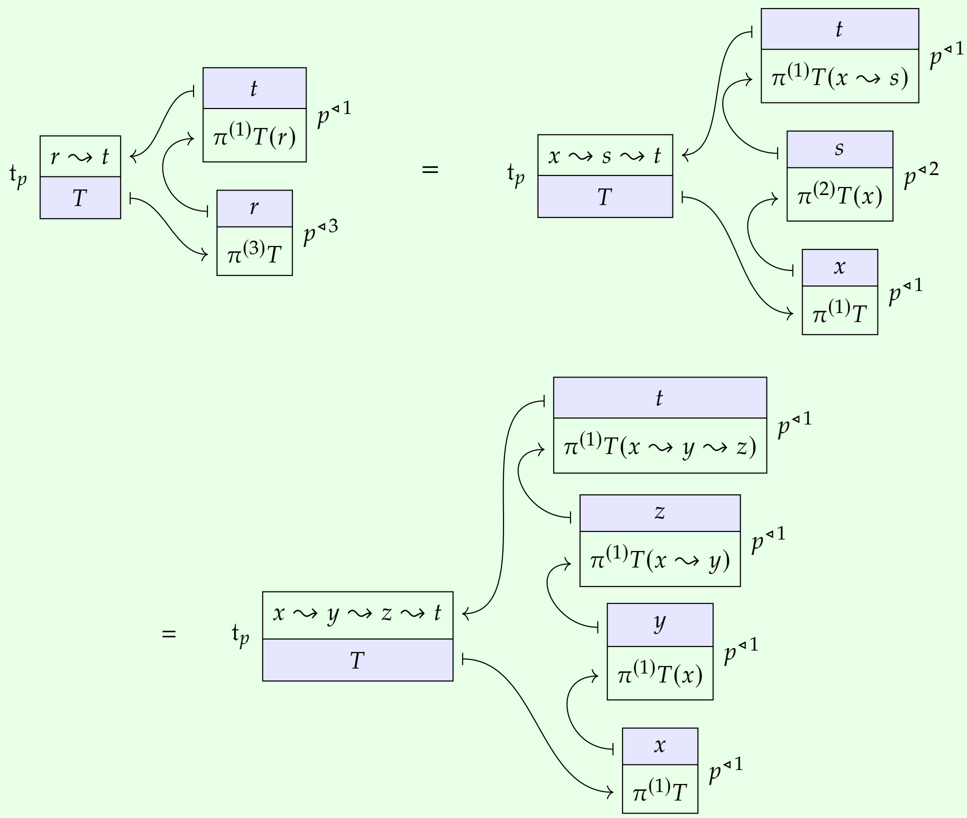


(8.25)

Here $T(u)$ denotes the p-tree equal to the subtree of T rooted at its vertex u, and w is one of $T(u)$'s height-2 vertices. When viewed as the subtree of T rooted at its the height-2 vertex u, the p-tree $T(u)$ has its height-2 vertex w identified with a height-4 vertex of T, a descendent of u that we denote by $u \rightsquigarrow w$. Alternatively, u corresponds to the rooted path (a_1, a_2) of T, while w corresponds to the rooted path (a_3, a_4) of $T(u)$, so $u \rightsquigarrow w$ corresponds to the concatenated rooted path (a_1, a_2, a_3, a_4) of T. As concatenation is associative, $\rightsquigarrow$ is associative as well.

Both the notation $T(u)$, for what we will call the *p-subtree* of the p-tree T rooted at $u \in \mathsf{vtx}(T)$, and the notation $u \rightsquigarrow w$, for the vertex in $\mathsf{vtx}(T)$ that $w \in \mathsf{vtx}(T(u))$ coincides with when $T(u)$ is identified with the subtree of p rooted at u, will turn out to be temporary—we will soon justify why we can express these concepts in much more familiar terms.[2] As a sneak preview, they will be crucial to defining the categorical structure of the cofree comonoid on p.

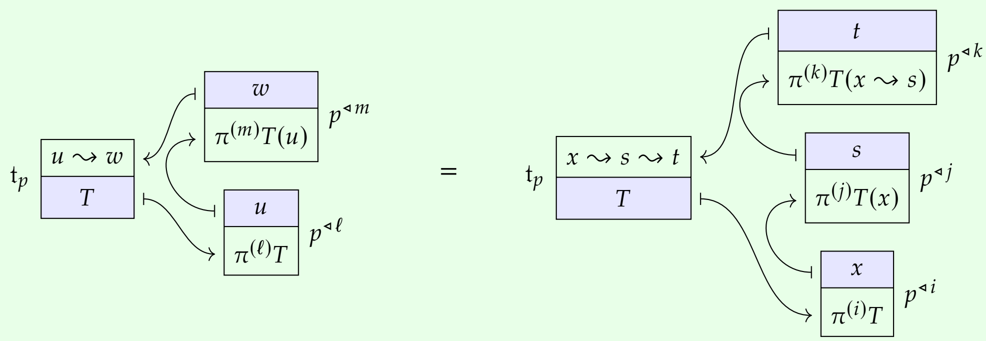
Here are three more ways to depict $\epsilon_p^{(4)}$, viewing its codomain in turn as $p^{\triangleleft 3} \triangleleft p^{\triangleleft 1}$, as $p^{\triangleleft 1} \triangleleft p^{\triangleleft 2} \triangleleft p^{\triangleleft 1}$, or as $p^{\triangleleft 1} \triangleleft p^{\triangleleft 1} \triangleleft p^{\triangleleft 1} \triangleleft p^{\triangleleft 1}$:



Here $r = x \rightsquigarrow s$ and $s = y \rightsquigarrow z$. Notice that the last depiction is just another way to write (8.22), with each $p^{\triangleleft 1}$-pretree in place of the p-position that labels its root and height-1 vertices in place of their corresponding directions, which are just length-1 rooted paths.

Finally, all this could be generalized to other values of n. Here are different ways to draw polyboxes for $\epsilon_p^{(n)} \colon \mathfrak{t}_p \to p^{\triangleleft n}$, viewing its codomain as $p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}$ with $n = \ell + m$

or $p^{\triangleleft i} \triangleleft p^{\triangleleft j} \triangleleft p^{\triangleleft k}$ with $n = i + j + k$:



8.1.2 Cofree comonoids as categories

We have now characterized the carrier $\mathfrak{t}_p$ of the comonoid $\mathscr{T}_p$ that will turn out to be cofree on p; but we have yet to describe the comonoid structure of $\mathscr{T}_p$ that $\mathfrak{t}_p$ carries. This structure will allow us to interpret $\mathscr{T}_p$ as a category, whose objects will be p-trees and whose morphisms will be the vertices of each p-tree. We could go ahead and describe this category right now, but as category theorists, let us show that the eraser and duplicator of $\mathscr{T}_p$, as well as the comonoid laws that they satisfy, arise naturally from our construction of $\mathfrak{t}_p$ as the limit of (8.1).

The eraser of the cofree comonoid

The eraser for a comonoid $\mathscr{T}_p$ carried by $\mathfrak{t}_p$ should be a lens of the form $\mathfrak{t}_p \to y$. Conveniently, since y appears in (8.1), its limit $\mathfrak{t}_p$ is already equipped with a canonical lens $\epsilon_p \colon \mathfrak{t}_p \to y$, as seen in (8.19). This lens will turn out to be the eraser of the cofree comonoid on p.

The eraser picks out a direction at each position of the carrier to be the identity morphism on that object. As each $\mathfrak{t}_p$-position is a tree $T \in \mathsf{tree}_p$, and each $\mathfrak{t}_p[T]$-direction is a vertex of T, the eraser ϵ_p should pick out a single vertex of every p-tree. Even without looking at (8.19), you may already have your suspicions as to which vertex this will turn out to be, as there is, after all, only one sensible way to choose a canonical vertex that every p-tree is guaranteed to have: choose its root. Indeed, Proposition 8.18 tells us that $\epsilon_p \colon \mathfrak{t}_p \to y$ sends each p-tree $T \in \mathsf{tree}_p$ to its stage-0 pretree on positions, stripping away everything except for the single unlabeled root of T; then on-directions, ϵ_p picks out the unique height-0 vertex of T in $\mathsf{vtx}_0(T) \subseteq \mathsf{vtx}(T)$, which is just its root.

So in the category $\mathscr{T}_p$, where morphisms out of objects are vertices of trees, the identity morphism on a tree is its root—in fact, we will henceforth denote the root of a p-tree by id_T (so that $\mathsf{vtx}_0(T) = \{\mathrm{id}_T\}$). Equivalently, we can identify id_T with the

[2] As a reminder, due to the definition of a p-tree T, a p-subtree of T is still itself a p-tree, whereas the $p^{\triangleleft n}$-pretree of T is not a p-tree unless the height of T is strictly less than n.

unique length-0 rooted path of T: the *empty rooted path*, which starts and ends at the root. The rest of the categorical structure of $\mathscr{T}_p$ will be determined by its duplicator.

The duplicator of the cofree comonoid

The duplicator for a comonoid $\mathscr{T}_p$ carried by $\mathfrak{t}_p$ should be a lens $\delta_p \colon \mathfrak{t}_p \to \mathfrak{t}_p \triangleleft \mathfrak{t}_p$. But before we can specify such a lens, we need to figure out what kind of polynomial $\mathfrak{t}_p \triangleleft \mathfrak{t}_p$ is.

Here is where our work from Sections 6.3.2 and 6.3.3 comes in handy: since the diagram in (8.1) is connected, its limit $\mathfrak{t}_p$ is a connected limit, and we showed in Theorem 6.80 that $\triangleleft$ preserves connected limits. Therefore $\mathfrak{t}_p \triangleleft \mathfrak{t}_p$ is itself the limit of some diagram, and a morphism to $\mathfrak{t}_p \triangleleft \mathfrak{t}_p$ is just a cone over that diagram.

Which diagram is it? First, we can use the fact that $\triangleleft$ preserves connected limits on the left to expand $\mathfrak{t}_p \triangleleft \mathfrak{t}_p$ as the limit of the following diagram, where we have applied the functor $- \triangleleft \mathfrak{t}_p$ to the diagram from (8.1):

$$\begin{tikzcd}
y \triangleleft \mathfrak{t}_p \arrow[d] & p \triangleleft \mathfrak{t}_p \arrow[d] & p^{\triangleleft 2} \triangleleft \mathfrak{t}_p \arrow[d] & p^{\triangleleft 3} \triangleleft \mathfrak{t}_p \arrow[d] & \cdots \\
1 \triangleleft \mathfrak{t}_p & p \triangleleft 1 \triangleleft \mathfrak{t}_p \arrow[l] & p^{\triangleleft 2} \triangleleft 1 \triangleleft \mathfrak{t}_p \arrow[l] & p^{\triangleleft 3} \triangleleft 1 \triangleleft \mathfrak{t}_p \arrow[l] & \cdots . \arrow[l]
\end{tikzcd}$$

But we have $p^{\triangleleft n} \triangleleft 1 \triangleleft \mathfrak{t}_p \cong p^{\triangleleft n} \triangleleft 1$, so we can simplify this diagram like so:

$$\begin{tikzcd}
y \triangleleft \mathfrak{t}_p \arrow[d] & p \triangleleft \mathfrak{t}_p \arrow[d] & p^{\triangleleft 2} \triangleleft \mathfrak{t}_p \arrow[d] & p^{\triangleleft 3} \triangleleft \mathfrak{t}_p \arrow[d] & \cdots \\
1 & p \triangleleft 1 \arrow[l] & p^{\triangleleft 2} \triangleleft 1 \arrow[l] & p^{\triangleleft 3} \triangleleft 1 \arrow[l] & \cdots . \arrow[l]
\end{tikzcd} \tag{8.26}$$

(This works because according to Proposition 6.68, $\triangleleft$ actually preserves *all* limits on the left—including the terminal object 1.) So $\mathfrak{t}_p \triangleleft \mathfrak{t}_p$ is the limit of (8.26).

Then we use the fact that $\triangleleft$ preserves connected limits on the right to expand each $p^{\triangleleft \ell} \triangleleft \mathfrak{t}_p$ in (8.26) as the limit of the following:

$$\begin{tikzcd}
p^{\triangleleft \ell} \triangleleft y \arrow[d] & p^{\triangleleft \ell} \triangleleft p \arrow[d] & p^{\triangleleft \ell} \triangleleft p^{\triangleleft 2} \arrow[d] & p^{\triangleleft \ell} \triangleleft p^{\triangleleft 3} \arrow[d] & \cdots \\
p^{\triangleleft \ell} \triangleleft 1 & p^{\triangleleft \ell} \triangleleft p \triangleleft 1 \arrow[l] & p^{\triangleleft \ell} \triangleleft p^{\triangleleft 2} \triangleleft 1 \arrow[l] & p^{\triangleleft \ell} \triangleleft p^{\triangleleft 3} \triangleleft 1 \arrow[l] & \cdots . \arrow[l]
\end{tikzcd} \tag{8.27}$$

So the limit of (8.26) is the limit of a larger diagram, where we have "plugged in" a copy of (8.27) in place of each $p^{\triangleleft \ell} \triangleleft \mathfrak{t}_p$ that appears.

It's worth being a little careful, though, when we draw this diagram: each $p^{\triangleleft \ell} \triangleleft \mathfrak{t}_p$ in (8.26) appears with a lens to $p^{\triangleleft \ell} \triangleleft 1$, and we need to work out what arrow should go in its place. The lens in question is given by applying $p^{\triangleleft \ell} \triangleleft -$ to the unique lens $\mathfrak{t}_p \to 1$. But 1 actually shows up in the lower left corner of (8.1), so the unique lens $\mathfrak{t}_p \to 1$ is just the projection from the limit of (8.1) to that 1. Then once we apply the connected limit-preserving functor $p^{\triangleleft \ell} \triangleleft -$ to (8.1), yielding (8.27), we obtain the lens $p^{\triangleleft \ell} \triangleleft \mathrm{t}_p \to p^{\triangleleft \ell} \triangleleft 1$ we desire as the projection from the limit of (8.27) to the $p^{\triangleleft \ell} \triangleleft 1$ in the lower left corner. So if we want to replace each $p^{\triangleleft \ell} \triangleleft \mathrm{t}_p$ in (8.26) with a copy of (8.27), without changing the limit of the whole diagram, we should replace the arrow $p^{\triangleleft \ell} \triangleleft \mathrm{t}_p \to p^{\triangleleft \ell} \triangleleft 1$ with an identity arrow from the $p^{\triangleleft \ell} \triangleleft 1$ in the lower left corner of (8.27) to the $p^{\triangleleft \ell} \triangleleft 1$ in (8.26). Of course, we can then collapse these identity arrows down without changing the limit, so the diagram we are left with should look like this:

$$
\begin{tikzcd}
p^{\triangleleft 3} \arrow[dr] & & p^{\triangleleft 3} \triangleleft p \arrow[dr] & & p^{\triangleleft 3} \triangleleft p^{\triangleleft 2} \arrow[dr] & & \cdots & \\
& p^{\triangleleft 3} \triangleleft 1 \arrow[dd] & & p^{\triangleleft 3} \triangleleft p \triangleleft 1 \arrow[dd] & & p^{\triangleleft 3} \triangleleft p^{\triangleleft 2} \triangleleft 1 \arrow[dd] & & \cdots \\
p^{\triangleleft 2} \arrow[dr] & & p^{\triangleleft 2} \triangleleft p \arrow[dr] & & p^{\triangleleft 2} \triangleleft p^{\triangleleft 2} \arrow[dr] & & \cdots & \\
& p^{\triangleleft 2} \triangleleft 1 \arrow[dd] & & p^{\triangleleft 2} \triangleleft p \triangleleft 1 \arrow[dd] & & p^{\triangleleft 2} \triangleleft p^{\triangleleft 2} \triangleleft 1 \arrow[dd] & & \cdots \\
p \arrow[dr] & & p \triangleleft p \arrow[dr] & & p \triangleleft p^{\triangleleft 2} \arrow[dr] & & \cdots & \\
& p \triangleleft 1 \arrow[dd] & & p \triangleleft p \triangleleft 1 \arrow[dd] & & p \triangleleft p^{\triangleleft 2} \triangleleft 1 \arrow[dd] & & \cdots \\
y \arrow[dr] & & p \arrow[dr] & & p^{\triangleleft 2} \arrow[dr] & & \cdots & \\
& 1 & & p \triangleleft 1 \arrow[ll] & & p^{\triangleleft 2} \triangleleft 1 \arrow[ll] & & \cdots . \arrow[ll]
\end{tikzcd}
\tag{8.28}
$$

Here the bottom row of (8.26) is still the bottom row of (8.28), but in the place of each $p^{\triangleleft \ell} \triangleleft \mathrm{t}_p$ in the top row, we have grafted in a copy of (8.27). Yet the limit of the diagram is preserved: the limit of (8.28) is still $\mathrm{t}_p \triangleleft \mathrm{t}_p$. In summary, for purely formal reasons, Proposition 6.68 and Theorem 6.80 ensure that since the limit of (8.1) is t_p, the limit of (8.28) is $\mathrm{t}_p \triangleleft \mathrm{t}_p$.

While (8.28) has become rather unwieldy to draw, it is easy to characterize: it is a diagram with a copy each of

$$
p^{\triangleleft \ell} \triangleleft p^{\triangleleft m} \qquad \text{and} \qquad p^{\triangleleft \ell} \triangleleft p^{\triangleleft m} \triangleleft 1
$$

for every $\ell, m \in \mathbb{N}$, with arrows

$$
p^{\triangleleft \ell} \triangleleft p^{\triangleleft m} \to p^{\triangleleft \ell} \triangleleft p^{\triangleleft m} \triangleleft 1 \tag{8.29}
$$

between them; along with arrows

$$
p^{\triangleleft \ell} \triangleleft p^{\triangleleft (m+1)} \triangleleft 1 \to p^{\triangleleft \ell} \triangleleft p^{\triangleleft m} \triangleleft 1, \tag{8.30}
$$

drawn vertically in (8.28); and

$$
p^{\triangleleft \ell+1} \triangleleft p^{\triangleleft 0} \triangleleft 1 \cong p^{\triangleleft \ell+1} \triangleleft 1 \to p^{\triangleleft \ell} \triangleleft 1 \cong p^{\triangleleft \ell} \triangleleft p^{\triangleleft 0} \triangleleft 1, \tag{8.31}
$$

drawn horizontally in (8.28) along the bottom row. Of course, we could write each $p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}$ as $p^{\triangleleft(\ell+m)}$, but the notation $p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}$ helps us distinguish it as the object appearing ℓ rows above the bottom and m columns to the right in (8.28), in contrast with any other $p^{\triangleleft \ell'} \triangleleft p^{\triangleleft m'}$ with $(\ell, m) \neq (\ell', m')$, even if $\ell + m = \ell' + m'$ guarantees that there are isomorphisms between these objects.

Nevertheless, it is these isomorphisms that induce a canonical lens $\mathsf{t}_p \to \mathsf{t}_p \triangleleft \mathsf{t}_p$, by giving a map from the diagram (8.1) to the diagram (8.28) as follows. For all $\ell, m, n \in \mathbb{N}$ with $n = \ell + m$, we have a canonical isomorphism $p^{\triangleleft n} \xrightarrow{\cong} p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}$ sending the $p^{\triangleleft n}$ that appears in (8.1) to the $p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}$ that appears in (8.28), and similarly sending $p^{\triangleleft n} \triangleleft 1$ in (8.1) to $p^{\triangleleft \ell} \triangleleft p^{\triangleleft m} \triangleleft 1$ in (8.28). Then all the arrows that appear in (8.28)—i.e. all the arrows in (8.29), (8.30), and (8.31)—can be identified with arrows that appear in (8.1), so everything commutes. We therefore induce a lens from the limit t_p of (8.1) to the limit $\mathsf{t}_p \triangleleft \mathsf{t}_p$ of (8.28), which we call $\delta_p \colon \mathsf{t}_p \to \mathsf{t}_p \triangleleft \mathsf{t}_p$. This turns out to be the duplicator of $\mathscr{T}_p$.

How does δ_p behave in terms of p-trees? First, we characterize $\mathsf{t}_p \triangleleft \mathsf{t}_p$ and its projections to each $p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}$ in (8.28). Concretely, we know that a $\mathsf{t}_p \triangleleft \mathsf{t}_p$-position is just a p-tree $T \in \mathsf{tree}_p$ along with a p-tree $U(v) \in \mathsf{tree}_p$ associated with every vertex $v \in \mathsf{vtx}(T)$. Then a direction at that position is a choice of vertex $v \in \mathsf{vtx}(T)$ and another vertex $w \in \mathsf{vtx}(U(v))$ of the p-tree $U(v)$ associated with v. Each projection from $\mathsf{t}_p \triangleleft \mathsf{t}_p$ to $p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}$ in (8.28) is then the composition product of the projections $\epsilon_p^{(\ell)} \colon \mathsf{t}_p \to p^{\triangleleft \ell}$ and $\epsilon_p^{(m)} \colon \mathsf{t}_p \to p^{\triangleleft m}$, which by Example 8.20 we can draw in polyboxes like so:

$$\begin{tikzcd} & \mathsf{t}_p & p^{\triangleleft m} & \\ \mathsf{vtx}(-) & \boxed{w} & \boxed{w} \arrow[l, mapsto] & \mathsf{vtx}_m(-) \\ \mathsf{tree}_p & \boxed{U_v} \arrow[r, mapsto] & \boxed{\pi^{(m)}U_v} & p^{\triangleleft m}(1) \end{tikzcd}$$

$$\begin{tikzcd} \mathsf{vtx}(-) & \boxed{v} & \boxed{v} \arrow[l, mapsto] & \mathsf{vtx}_\ell(-) \\ \mathsf{tree}_p & \boxed{T} \arrow[r, mapsto] & \boxed{\pi^{(\ell)}T} & p^{\triangleleft \ell}(1) \\ & \mathsf{t}_p & p^{\triangleleft \ell} & \end{tikzcd}$$

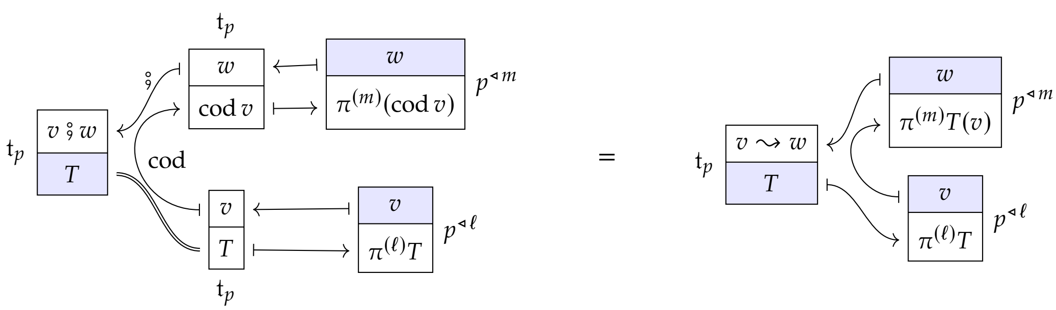
On positions, $\epsilon_p^{(\ell)} \triangleleft \epsilon_p^{(m)}$ sends the p-tree T to its stage-ℓ pretree $\pi^{(\ell)}T$, and it sends each p-tree U_v associated to a height-ℓ vertex $v \in \mathsf{vtx}_\ell(T)$ to its stage-m pretree $\pi^{(m)}U_v$, to be $p^{\triangleleft m}$-pretree associated with the height-ℓ leaf v of $\pi^{(m)}T$. Equivalently, this specifies a $p^{\triangleleft(\ell+m)}$-pretree on the right: its bottom ℓ levels coincide with the bottom ℓ levels of T, and its top m levels coincide with the bottom m levels of the U_v's for $v \in \mathsf{vtx}_\ell(T)$. Then on directions, $\epsilon_p^{(\ell)} \triangleleft \epsilon_p^{(m)}$ is the canonical inclusion of vertices $\mathsf{vtx}_\ell(T) \hookrightarrow \mathsf{vtx}(T)$ sending $v \mapsto v$, followed by the canonical inclusion of vertices $\mathsf{vtx}_m(U_v) \hookrightarrow \mathsf{vtx}(U_v)$. These lenses comprise the universal cone over (8.28).

Meanwhile, the other cone we formed over (8.28) is comprised of lenses of the form $\epsilon_p^{(\ell+m)} \colon \mathsf{t}_p \to p^{\triangleleft(\ell+m)} \cong p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}$, each of which should factor through $\mathsf{t}_p \triangleleft \mathsf{t}_p$, as depicted in the following commutative diagram:

$$\begin{tikzcd}
\mathsf{t}_p \arrow[r, dashed, "\delta_p"] \arrow[dr, "\epsilon_p^{(\ell+m)}"'] & \mathsf{t}_p \triangleleft \mathsf{t}_p \arrow[d, "\epsilon_p^{(\ell)} \triangleleft \epsilon_p^{(m)}"] \\
& p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}.
\end{tikzcd} \tag{8.32}$$

Indeed, by the universal property of $\mathsf{t}_p \triangleleft \mathsf{t}_p$, our δ_p is the unique lens for which the above diagram commutes for all $\ell, m \in \mathbb{N}$. We will use the equation given by the commutativity of (8.32) repeatedly in what follows, whenever we work with δ_p.

Expressing (8.32) as an equation of polyboxes, using our usual labels for the arrows of the duplicator δ_p on the left and our depiction of the projection $\epsilon_p^{(\ell+m)} \colon \mathsf{t}_p \to p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}$ from Example 8.20 on the right, we have



Recall from Example 8.20 that $T(v)$ denotes the p-tree equal to the subtree of T rooted at $v \in \mathsf{vtx}(T)$, while $v \rightsquigarrow w \in \mathsf{vtx}(T)$ for $w \in \mathsf{vtx}(T(v))$ is the descendent of v that coincides with w when $T(v)$ is identified with the subtree of T rooted at v. Equivalently, we can identify $v \in \mathsf{vtx}(T)$ with the rooted path in T that ends at the vertex v and $w \in \mathsf{vtx}(T(v))$ with the rooted path of $T(v)$ that ends at the vertex w, so $v \rightsquigarrow w$ becomes the rooted path in T obtained by concatenating v and w. Then for this equality to hold over all $\ell, m \in \mathbb{N}$, we want $\text{cod}\, v \coloneqq T(v)$ and $v \fatsemi w \coloneqq v \rightsquigarrow w$; in fact, $\text{cod}\, v$ and $v \fatsemi w$ will henceforth be our preferred notation for $T(v)$ and $v \rightsquigarrow w$.

Verifying the comonoid laws

Putting together our constructions of the carrier, the eraser, and the duplicator of the $\mathscr{T}_p$ yields the following result.

Proposition 8.33. As defined above, $(\mathsf{t}_p, \epsilon_p, \delta_p)$ is a polynomial comonoid corresponding to a category $\mathscr{T}_p$ characterized as follows.

- An object in $\mathscr{T}_p$ is a *p-tree* in $T \in \mathsf{tree}_p$.
- A morphism emanating from T is a *rooted path* in T; its codomain is the p-subtree rooted at the end of the path.
- The identity morphism on T is its empty rooted path.
- Composition is given by concatenating rooted paths: given a rooted path v in T and a rooted path w in $\operatorname{cod} v$, the p-subtree rooted at the end of v, we identify w with the corresponding path in T that starts where v ends, then concatenate the two paths in T to obtain the composite morphism $v \fatsemi w$, another rooted path in T.

Proof. We have already shown how to construct the given carrier, eraser, and duplicator purely diagrammatically, and we have given them concrete interpretations in terms of p-trees, their vertices, and their p-subtrees. So it remains to verify that the category laws hold for $\mathscr{T}_p$, or equivalently that the comonoid laws hold for $(\mathsf{t}_p, \epsilon_p, \delta_p)$. Again, our argument will be purely formal, although it is not too hard to see that our concrete characterization above in terms of p-trees satisfies the laws for a category.

First, we verify the left erasure law: that

$$\begin{tikzcd}
y \triangleleft \mathsf{t}_p \arrow[r, equal] & \mathsf{t}_p \arrow[d, "\delta_p"] \\
& \mathsf{t}_p \triangleleft \mathsf{t}_p \arrow[ul, "\epsilon_p \triangleleft \mathsf{t}_p"]
\end{tikzcd}$$

commutes. We know the lens $\epsilon_p \triangleleft \mathsf{t}_p \colon \mathsf{t}_p \triangleleft \mathsf{t}_p \to y \triangleleft \mathsf{t}_p$ is characterized by its components $\epsilon_p \triangleleft \epsilon_p^{(n)} \colon \mathsf{t}_p \triangleleft \mathsf{t}_p \to p^{\triangleleft 0} \triangleleft p^{\triangleleft n}$, a projection out of the limit $\mathsf{t}_p \triangleleft \mathsf{t}_p$, for all $n \in \mathbb{N}$. Then by our construction of δ_p, each composite $\delta_p \fatsemi \left(\epsilon_p \triangleleft \epsilon_p^{(n)}\right) \colon \mathsf{t}_p \to p^{\triangleleft 0} \triangleleft p^{\triangleleft n}$ is the component of δ_p equal to $\epsilon_p^{(n)} = \epsilon_p^{(0+n)} \colon \mathsf{t}_p \to p^{\triangleleft 0} \triangleleft p^{\triangleleft n} \cong p^{\triangleleft n}$ (this is just the commutativity of (8.32) in the case of $(\ell, m) = (0, n)$). Together, these characterize the composite lens $\delta_p \fatsemi (\epsilon_p \triangleleft \mathsf{t}_p) \colon \mathsf{t}_p \to y \triangleleft \mathsf{t}_p \cong \mathsf{t}_p$ as the lens whose components are the projections $\epsilon_p^{(n)} \colon \mathsf{t}_p \to p^{\triangleleft n}$ from the limit. It follows from the universal property of t_p that $\delta_p \fatsemi (\epsilon_p \triangleleft \mathsf{t}_p)$ can only be the identity lens on t_p. Hence the left erasure law holds.

The right erasure law, that

$$\begin{tikzcd}
\mathsf{t}_p \arrow[r, equal] \arrow[d, "\delta_p"'] & \mathsf{t}_p \triangleleft y \\
\mathsf{t}_p \triangleleft \mathsf{t}_p \arrow[ur, "\mathsf{t}_p \triangleleft \epsilon_p"']
\end{tikzcd}$$

commutes, follows similarly: the composite $\delta_p \fatsemi (\mathsf{t}_p \triangleleft \epsilon_p)$ is characterized by its components over $n \in \mathbb{N}$ of the form $\delta_p \fatsemi \left(\epsilon_p^{(n)} \triangleleft \epsilon_p\right) \colon \mathsf{t}_p \to p^{\triangleleft n} \triangleleft p^{\triangleleft 0}$, which we know by (8.32) is equal to $\epsilon_p^{(n+0)} = \epsilon_p^{(n)}$, the projection out of the limit t_p. It follows from the universal property of this limit that $\delta_p \fatsemi (\mathsf{t}_p \triangleleft \epsilon_p)$ must be the identity lens on t_p.

Finally, we check the coassociative law: that

$$\begin{tikzcd} \mathfrak{t}_p \arrow[r, "\delta_p"] \arrow[d, "\delta_p"'] & \mathfrak{t}_p \triangleleft \mathfrak{t}_p \arrow[d, "\mathfrak{t}_p \triangleleft \delta_p"] \\ \mathfrak{t}_p \triangleleft \mathfrak{t}_p \arrow[r, "\delta_p \triangleleft \mathfrak{t}_p"'] & \mathfrak{t}_p \triangleleft \mathfrak{t}_p \triangleleft \mathfrak{t}_p, \end{tikzcd}$$

commutes. Because $\triangleleft$ preserves connected limits, we can write $\mathfrak{t}_p \triangleleft \mathfrak{t}_p \triangleleft \mathfrak{t}_p$ as a limit the way we did with $\mathfrak{t}_p \triangleleft \mathfrak{t}_p$: it is the limit of diagram consisting of arrows

$$p^{\triangleleft i} \triangleleft p^{\triangleleft j} \triangleleft p^{\triangleleft k} \to p^{\triangleleft i} \triangleleft p^{\triangleleft j} \triangleleft p^{\triangleleft k} \triangleleft 1$$

for each $i, j, k \in \mathbb{N}$, with additional arrows between the position-sets. So a lens to $\mathfrak{t}_p \triangleleft \mathfrak{t}_p \triangleleft \mathfrak{t}_p$, such as those in the square above, is uniquely determined by its components to $p^{\triangleleft i} \triangleleft p^{\triangleleft j} \triangleleft p^{\triangleleft k}$ for all $i, j, k \in \mathbb{N}$, obtained by composing it with the projections $\epsilon_p^{(i)} \triangleleft \epsilon_p^{(j)} \triangleleft \epsilon_p^{(k)}$ out of the limit. Then by (8.32),

$$\begin{aligned} \delta_p \mathbin{⨟} (\mathfrak{t}_p \triangleleft \delta_p) \mathbin{⨟} \left(\epsilon_p^{(i)} \triangleleft \epsilon_p^{(j)} \triangleleft \epsilon_p^{(k)}\right) &= \delta_p \mathbin{⨟} \left(\epsilon_p^{(i)} \triangleleft \left(\delta_p \mathbin{⨟} \left(\epsilon_p^{(j)} \triangleleft \epsilon_p^{(k)}\right)\right)\right) \\ &= \delta_p \mathbin{⨟} \left(\epsilon_p^{(i)} \triangleleft \epsilon_p^{(j+k)}\right) && (8.32) \\ &= \epsilon_p^{(i+j+k)} && (8.32) \\ &= \delta_p \mathbin{⨟} \left(\epsilon_p^{(i+j)} \triangleleft \epsilon_p^{(k)}\right) && (8.32) \\ &= \delta_p \mathbin{⨟} \left(\left(\delta_p \mathbin{⨟} \left(\epsilon_p^{(i)} \triangleleft \epsilon_p^{(j)}\right)\right) \triangleleft \epsilon_p^{(k)}\right) && (8.32) \\ &= \delta_p \mathbin{⨟} (\delta_p \triangleleft \mathfrak{t}_p) \mathbin{⨟} \left(\epsilon_p^{(i)} \triangleleft \epsilon_p^{(j)} \triangleleft \epsilon_p^{(k)}\right). \end{aligned}$$

for all $i, j, k \in \mathbb{N}$. So by the universal property of $\mathfrak{t}_p \triangleleft \mathfrak{t}_p \triangleleft \mathfrak{t}_p$, coassociativity holds. □

We call the category $\mathscr{T}_p$ corresponding to the comonoid $(\mathfrak{t}_p, \epsilon_p, \delta_p)$ the *category of p-trees*, the *category of trees on p*, or the *p-tree category*.

Example 8.34 (The category of p-trees: states and transitions for (co)free). Let's step back and think about how p-tree categories relate to the original polynomial p from the perspective of the states and transitions of dynamical systems. Before we build any trees or categories out of it, a polynomial p is just a family of sets of directions indexed over a set of positions. The directions emerge from positions, but don't point to anywhere in particular. If we want to interpret the positions of p as states and the directions of p as composable transitions, first we would need to point the directions to other positions by assigning them codomains. There are many ways to do this, but we would like to do so "freely," without having to make any choices along the way that would bias us one way or another, so as to give a canonical way to interpret p as a category.

So to avoid making choices, rather than fixing a codomain for each direction, we allow every possible direction to point to every possible position once. But a single direction of any one position cannot point to multiple positions, which is when we need to make several copies of each position: at least one copy for every possible combination of codomains that can be assigned to its directions. This is how we get from p to $p^{\triangleleft 2}$: the $p^{\triangleleft 2}$-pretrees represent all the ways we could assign codomains to the directions in p. Essentially, we have freely refined our positions into more specific states to account for everywhere their directions could lead. Each $p^{\triangleleft 2}$-pretree still remembers which $p^{\triangleleft 1}$-pretree it grew from, giving us our canonical trimming operation $p^{\triangleleft 2}(1) \to p^{\triangleleft 1}(1)$.

Yet even that is not enough: sure, we've assigned codomains to the directions of p in every possible way, but now that we're building a category, we want our directions to be composable transitions. So each pair of composable directions of p—each length-2 rooted path of the $p^{\triangleleft 2}$-pretrees—is now a possible transition as well, another morphism in our category. To keep everything canonical, we still want to avoid making any actual choices; we can't simply say that two directions of p compose to a third direction of p that we already have. Each pair of composable directions must be an entirely new morphism—and every new morphism needs a codomain.

You can tell where this is going. To avoid actually choosing codomains for the new morphisms, we need to refine our $p^{\triangleleft 2}$-pretrees by making copies of them to account for every possibility, building $p^{\triangleleft 3}$-pretrees as a result. Then their length-3 rooted paths are new morphisms, too, and they need new codomains, and so forth, ad infinitum. Indeed, this process cannot terminate in a finite number of steps, but that's okay—we can take the limit, yielding the ultimate free refinement of positions into states and free composites of directions as transitions.

This is where (8.1) comes from; it captures the infinite process of turning sequences of directions into morphisms by giving them codomains in every possible way, then dealing with the longer sequences of directions that emerge as a result. Note that we also need to account for the fact that every object needs an identity; to avoid making a choice, we don't set it to be any direction or composite of multiple directions of p, reserving it instead for the empty sequence of directions that every pretree has. Then the limit $\mathfrak{t}_p$ of (8.1) is all we need: the p-trees are states representing every possible way that sequences of directions emerging from a p-position can lead to other p-positions, and the rooted paths of each p-tree are transitions accounting for every finite sequence of composable directions from the corresponding state.

Since composites were freely generated, directions and thus entire rooted paths compose by concatenation—making the empty rooted path the correct identity. And the codomain is whatever the direction at the end of a rooted path has been freely assigned to point to—not just the p-position there but the whole p-subtree, an entire state representing all the sequences of directions one could take and the positions to which they lead, starting from the end of the path just followed. This gives the comonoid structure on $\mathfrak{t}_p$, completing the cofree construction of the category of p-trees $\mathscr{T}_p$.

Soon we will prove that $\mathscr{T}_p$ really is the cofree category on p, but first we give some examples to make all this concrete.

Examples of p-tree categories

In what follows, we will freely switch between the morphisms-as-vertices and the morphisms-as-rooted-paths perspectives of p-tree categories given in Proposition 8.33 whenever convenient.

Example 8.35 (The category of 1-trees). Let's start by taking $p := 1$. In Exercise 8.17 #1, we showed that there is a unique 1-tree: a tree with only 1 vertex, its root. So $\mathsf{t}_1 \cong y$ is the carrier of the category of 1-trees $\mathscr{T}_1$.

Up to isomorphism, there is only one category carried by y: the terminal category with 1 object and no morphisms aside from the 1 identity. When we think of this as the category of 1-trees, we can characterize it as follows.

- The 1 object is the tree with 1 vertex: call the tree $\bullet$, because that's what it looks like.
- The 1 morphism $\bullet \to \_$ is the 1 vertex of $\bullet$, its root. After all, there are no directions of 1 to freely compose, so the only morphism generated is the identity. Since the 1-subtree of $\bullet$ rooted as its root is just the entire 1-tree $\bullet$, the codomain of this morphism is still $\bullet$, which makes sense—it's the only object we have.
- The identity morphism $\mathrm{id}_\bullet \colon \bullet \to \bullet$ is the root of $\bullet$. This is the same morphism we just mentioned. Equivalently, it is the empty rooted path.
- The composite morphism $\mathrm{id}_\bullet \mathbin{⨟} \mathrm{id}_\bullet = \mathrm{id}_\bullet \mathbin{⨟} \mathrm{id}_{\mathrm{cod}(\mathrm{id}_\bullet)} \colon \bullet \to \_$ is obtained by concatenating the empty rooted path $\mathrm{id}_\bullet$ of $\bullet$ with the empty rooted path $\mathrm{id}_{\mathrm{cod}(\mathrm{id}_\bullet)}$ of $\mathrm{cod}(\mathrm{id}_\bullet)$, which is again just the empty rooted path of $\bullet$. Hence $\mathrm{id}_\bullet \mathbin{⨟} \mathrm{id}_{\mathrm{cod}\,\mathrm{id}_\bullet} = \mathrm{id}_\bullet$, as expected—it's the only morphism we have.

This is certainly too trivial an example to say much about, but it demonstrates how we can interpret the category carried by y as the category of p-trees for $p := 1$. Moreover, it helps us see concretely why taking the identity to be the root gives it the right codomain and compositional behavior.

Exercise 8.36 (The category of trees on a constant; solution here). Let B be a set, viewed as a constant polynomial.

1. What is the polynomial t_B?
2. Characterize the B-tree category $\mathscr{T}_B$. ◊

Example 8.37 (The category of y-trees). Now consider $p := y$. In Exercise 8.17 #3, we showed that there is a unique y-tree: a single ray extending from the root in which every vertex has exactly 1 child, so that there is exactly 1 height-n vertex—and 1 length-n rooted path—for every $n \in \mathbb{N}$. Then $\mathsf{t}_y \cong y^{\mathbb{N}}$ is the carrier of the category of y-trees

$\mathscr{T}_y$. In fact, we can identify the set of rooted paths of this y-tree with $\mathbb{N}$, so that $n \in \mathbb{N}$ is the y-tree's unique length-n rooted path.

We know from Example 7.40 that comonoids with representable carriers correspond to 1-object categories, which can be identified with monoids. In particular, a comonoid structure on $y^{\mathbb{N}}$ corresponds to a monoid structure on $\mathbb{N}$. There is more than one monoid structure on $\mathbb{N}$, though, so which one corresponds to the category $\mathscr{T}_y$?

We can characterize $\mathscr{T}_y$ in terms of y-trees as follows.

- The 1 object is the unique y-tree, a single ray: call it $\uparrow$. Here is a picture of this ray, to help you visualize its vertices, rooted paths, and subtrees:

- The morphisms are the rooted paths of $\uparrow$; they comprise the set $\mathbb{N}$, where each $n \in \mathbb{N}$ is the unique rooted path of $\uparrow$ of length n. Each rooted path represents the free composite of n copies of the sole direction of y. Since $\uparrow$ is an infinite ray, the y-subtree of $\uparrow$ rooted at any of its vertices is still just a copy of $\uparrow$. So the codomain of each morphism is still $\uparrow$, which makes sense—it's the only object we have.
- The identity morphism $\mathrm{id}_{\uparrow} \colon \uparrow \to \uparrow$ is the empty rooted path, which has length 0; so $\mathrm{id}_{\uparrow} = 0$. In the corresponding monoid structure on $\mathbb{N}$, the element $0 \in \mathbb{N}$ must then be the unit.
- The composite morphism $m \mathbin{⨟} n \colon \uparrow \to \uparrow$ for $m, n \in \mathbb{N}$ is obtained by concatenating m, i.e. the length-m rooted path of $\uparrow$, with n, i.e. the length-n rooted path of $\uparrow$, translating the latter path so that it begins where the former path ends. The result is then the rooted path of $\uparrow$ of length $m + n$; so $m \mathbin{⨟} n = m + n$. In the corresponding monoid structure on $\mathbb{N}$, the binary operation must then be given by addition.

Hence $\mathscr{T}_y$ is the monoid $(\mathbb{N}, 0, +)$ viewed as a 1-object category.

Example 8.38 (By-trees are B-streams). Let B be a set, and consider $p := By$. Generalizing Exercise 8.17 #5 for B instead of $\mathsf{2}$, or applying Exercise 8.17 #7 for the case of $A := \mathsf{1}$, we can deduce that a By-tree consists of a single ray for which every vertex is given a label from B. So the vertices of a By-tree are in bijection with $\mathbb{N}$, and By-trees are in bijection with functions $\mathbb{N} \to B$ assigning each vertex a label. Hence $\mathfrak{t}_{By} \cong B^{\mathbb{N}} y^{\mathbb{N}}$ is the carrier of the category of By-trees $\mathscr{T}_{By}$. As in Example 8.37, we identify the set of rooted paths of a given By-tree with $\mathbb{N}$, so that $n \in \mathbb{N}$ is the By-tree's unique length-n rooted path.

In fact, we have already seen the category $\mathscr{T}_{By}$ once before: it is the category of B-streams from Example 7.45.

- Recall that a B-stream is an element of $B^{\mathbb{N}}$ interpreted as a countable sequence of elements $b_n \in B$ for $n \in \mathbb{N}$, written like so:

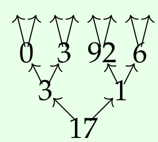
$$\overline{b} := (b_0 \to b_1 \to b_2 \to b_3 \to \cdots).$$

  This is just a By-tree lying on its side! Each arrow is a copy of the sole direction at each position of By, pointing to one of the positions in B for that direction to lead to next.
- Given a B-stream $\overline{b} \in B^{\mathbb{N}}$ above, recall that a morphism out of $\overline{b}$ is a natural number $n \in \mathbb{N}$, and its codomain is the substream of B starting at b_n:

$$\operatorname{cod}(\overline{b} \xrightarrow{n} \_) = (b_n \to b_{n+1} \to b_{n+2} \to b_{n+3} \to \cdots).$$

  This coincides with how we view morphisms as rooted paths and codomains as subtrees rooted at the end of those paths in the category of By-trees.
- Whether we view $\overline{b}$ as a B-stream or a By-tree, its identity morphism $\mathrm{id}_{\overline{b}} \colon \overline{b} \to \overline{b}$ corresponds to $0 \in \mathbb{N}$; from the latter perspective, it is the length-0 path from the root to itself.
- Whether we view $\overline{b}$ as a B-stream or a By-tree, composition is given by addition; from the latter perspective, concatenating a length-m rooted path with a length-n path yields a length-$(m+n)$ rooted path.

Example 8.39 (The category of $\mathbb{N}$-labeled binary trees). Consider $p := \mathbb{N}y^2$. By Exercise 8.17 #7, or Exercise 8.12 #2 in the case of $L := \mathbb{N}$ and $\mathsf{n} := \mathsf{2}$, an $\mathbb{N}y^2$-tree is an infinite binary tree with vertices labeled by elements of $\mathbb{N}$. Here's how such an $\mathbb{N}y^2$-tree might start:



We can characterize the category $\mathscr{T}_{\mathbb{N}y^2}$ as follows.

- The objects are $\mathbb{N}$-labeled binary trees from the set $\mathbb{N}^{\mathsf{List}(\mathsf{2})}$.
- A morphism out of an $\mathbb{N}$-labeled binary tree is a binary sequence: a finite list of directions in $\mathsf{2}$, whose elements could be interpreted as "left" and "right," thus uniquely specifying a rooted path in a binary tree. They comprise the set $\mathsf{List}(\mathsf{2})$. The codomain of each rooted path is the $\mathbb{N}$-labeled binary subtree rooted at the end of the path.
- The identity morphism on a given $\mathbb{N}$-labeled binary tree is its empty rooted path.
- The composite of two binary sequences is obtained by concatenation.

Exercise 8.40 (The category of B-labeled A-ary trees; solution here). Characterize the By^A-tree category $\mathscr{T}_{By^A}$. ◊

Exercise 8.41 (Solution here). Characterize the $(y + 1)$-tree category $\mathscr{T}_{y+1}$. ◊

Exercise 8.42 (Solution here). Let $p := \{a, b, c, \ldots, z, \sqcup\}y + \{\bullet\}$.

1. Describe the objects of the cofree category $\mathscr{T}_p$, and draw one.
2. For a given such object, describe the set of emanating morphisms.
3. Describe how to take the codomain of a morphism. ◊

Exercise 8.43 (Solution here). Let $p := \{\bullet, \bullet\}y^2 + \{\bullet\}y + \{\bullet\}$ as in Example 8.10.

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. ◊

Exercise 8.44 (Solution here). Let p be a polynomial, let $Q := \{q \in \mathbb{Q} \mid q \geq 0\}$ and consider the monoid y^Q of nonnegative rational numbers under addition. Is it true that any retrofunctor $\varphi \colon \mathscr{T}_p \nrightarrow y^{\mathbb{Q}}$ is constant, i.e. that it factors as

$$\mathscr{T}_p \nrightarrow y \nrightarrow y^{\mathbb{Q}}?$$

◊

8.1.3 Exhibiting the forgetful-cofree adjunction

We are now ready to give a diagrammatic proof of the main result of this section: as promised, the category $\mathscr{T}_p$ we constructed is the cofree comonoid on p.

Theorem 8.45 (Cofree comonoid)**.** The forgetful functor $U\colon \mathbf{Cat}^\sharp \to \mathbf{Poly}$ has a right adjoint $\mathscr{T}_-\colon \mathbf{Poly} \to \mathbf{Cat}^\sharp$, giving rise to an adjunction

$$\begin{tikzcd} \mathbf{Cat}^\sharp \arrow[r, shift left, "U"] \arrow[r, phantom, "\Rightarrow"] & \mathbf{Poly} \arrow[l, shift left, "\mathscr{T}_-"] \end{tikzcd},$$

such that for each $p \in \mathbf{Poly}$, the carrier $\mathfrak{t}_p := U\mathscr{T}_p$ of the category $\mathscr{T}_p$ is given by the limit of the diagram (8.1), repeated here:

$$\begin{tikzcd} y \arrow[d] & p \arrow[d] & p^{\triangleleft 2} \arrow[d] & p^{\triangleleft 3} \arrow[d] & \cdots \\ 1 & p \triangleleft 1 \arrow[l, "!"'] & p^{\triangleleft 2} \triangleleft 1 \arrow[l, "p \triangleleft !"'] & p^{\triangleleft 3} \triangleleft 1 \arrow[l, "p^{\triangleleft 2} \triangleleft !"'] & \cdots \arrow[l] \end{tikzcd}.$$

That is, for any category $\mathcal{C} \in \mathbf{Cat}^\sharp$ with carrier $\mathfrak{c} := U\mathcal{C}$, there is a natural isomorphism

$$\mathbf{Poly}(\mathfrak{c}, p) \cong \mathbf{Cat}^\sharp(\mathcal{C}, \mathscr{T}_p).$$

Proof. To show that $\mathscr{T}_-$ is the right adjoint of U, it is enough to show that for every lens $\varphi\colon \mathfrak{c} \to p$, there exists a unique retrofunctor $F\colon \mathcal{C} \nrightarrow \mathscr{T}_p$ for which

$$\begin{tikzcd} \mathfrak{c} \arrow[d, dashed, "F"'] \arrow[dr, "\varphi"] & \\ \mathfrak{t}_p \arrow[r, "\epsilon_p^{(1)}"'] & p \end{tikzcd} \tag{8.46}$$

commutes. Here the projection $\epsilon_p^{(1)}\colon U\mathscr{T}_p \cong \mathfrak{t}_p \to p$ serves as the counit of the adjunction, and we identify the retrofunctor F with its underlying lens $UF\colon \mathfrak{c} \to \mathfrak{t}_p$.

First, we construct F from φ as follows. If we let ϵ and δ be the eraser and duplicator of $\mathcal{C}$, the diagram

$$\begin{tikzcd} \mathfrak{c} \arrow[r, "\delta"] \arrow[d, "\epsilon"'] \arrow[dr, "\varphi"] & \mathfrak{c} \triangleleft \mathfrak{c} \arrow[d, "\epsilon \triangleleft \varphi"] \\ y \arrow[d] & p \arrow[d] \\ 1 & p \triangleleft 1 \arrow[l] \end{tikzcd} \tag{8.47}$$

commutes: the pentagon in the lower left commutes trivially, while the triangle in the upper right commutes by the left erasure law of $\mathcal{C}$, as

$$\begin{aligned} \delta \mathbin{⨟} (\epsilon \triangleleft \varphi) &= \delta \mathbin{⨟} (\epsilon \triangleleft \mathfrak{c}) \mathbin{⨟} \varphi \\ &= \mathrm{id}_{\mathfrak{c}} \mathbin{⨟} \varphi && \text{(Left erasure law)} \\ &= \varphi. \end{aligned}$$

Then by induction, the larger diagram

$$
\begin{tikzcd}
\mathfrak{c} \arrow[r, "\delta"] \arrow[d, "\epsilon"'] \arrow[dr, "\varphi"'] & \mathfrak{c} \triangleleft \mathfrak{c} \arrow[r, "\delta \triangleleft \mathfrak{c}"] \arrow[d, "\epsilon \triangleleft \varphi"] & \mathfrak{c}^{\triangleleft 3} \arrow[r, "\delta \triangleleft \mathfrak{c}^{\triangleleft 2}"] \arrow[d, "\epsilon \triangleleft \varphi^{\triangleleft 2}"] & \mathfrak{c}^{\triangleleft 4} \arrow[r] \arrow[d, "\epsilon \triangleleft \varphi^{\triangleleft 3}"] & \cdots \\
y \arrow[d] & p \arrow[d] & p^{\triangleleft 2} \arrow[d] & p^{\triangleleft 3} \arrow[d] & \cdots \\
1 & p \triangleleft 1 \arrow[l] & p^{\triangleleft 2} \triangleleft 1 \arrow[l] & p^{\triangleleft 3} \triangleleft 1 \arrow[l] & \cdots \arrow[l]
\end{tikzcd}
\tag{8.48}
$$

commutes as well: its leftmost rectangle is (8.47), and taking the composition product of each rectangle in (8.48) with the commutative rectangle

$$
\begin{tikzcd}
\mathfrak{c} \arrow[r, equal] \arrow[d, "\varphi"'] & \mathfrak{c} \arrow[d, "\varphi"] \\
p \arrow[d, equal] & p \arrow[d, equal] \\
p \arrow[r, equal] & p
\end{tikzcd}
$$

yields the rectangle to its right. As $\mathfrak{t}_p$ is the limit of the bottom two rows of (8.48), it follows that there is an induced lens $F\colon \mathfrak{c} \to \mathfrak{t}_p$ that, when composed with each projection $\epsilon_p^{(n)}\colon \mathfrak{t}_p \to p^{\triangleleft n}$, yields the lens depicted in (8.48) from $\mathfrak{c}$ to $p^{\triangleleft n}$. This lens is the composite of the lens $\mathfrak{c} \to \mathfrak{c}^{\triangleleft (n+1)}$ in the top row, which by Proposition 7.20 is the canonical lens $\delta^{(n+1)}$ associated with the comonoid $\mathcal{C}$, composed with the lens $\epsilon \triangleleft \varphi^{\triangleleft n}\colon \mathfrak{c}^{\triangleleft (n+1)} \to p^{\triangleleft n}$.

Next, we prove uniqueness: that a retrofunctor $\mathcal{C} \nrightarrow \mathscr{T}_p$ with underlying lens $f\colon \mathfrak{c} \to \mathfrak{t}_p$ is completely determined by the value of $f \mathbin{⨟} \epsilon_p^{(1)}$. It suffices to show that we can recover the n^{th} component of g from its first component. **

□

8.1.4 The many (inter)faces of the cofree comonoid

The forgetful-cofree adjunction of Theorem 8.45 tells us that given a category $\mathcal{C} \in \mathbf{Cat}^\sharp$ with carrier $\mathfrak{c} := U\mathcal{C}$ and a polynomial $p \in \mathbf{Poly}$, there is a natural isomorphism

$$\mathbf{Poly}(\mathfrak{c}, p) \cong \mathbf{Cat}^\sharp(\mathcal{C}, \mathscr{T}_p).$$

So every lens $\varphi\colon \mathfrak{c} \to p$ has a corresponding retrofunctor $F\colon \mathcal{C} \nrightarrow \mathscr{T}_p$ that we call its *mate*.

We can view φ as a dynamical system with an interface p and a generalized state system $\mathfrak{c}$, carrying an arbitrary category $\mathcal{C}$ of states and transitions. Then the lens $\mathrm{Run}_n(\varphi)\colon \mathfrak{c} \to p^{\triangleleft n}$ for $n \in \mathbb{N}$, defined as the composite

$$\mathfrak{c} \xrightarrow{\delta^{(n)}} \mathfrak{c}^{\triangleleft n} \xrightarrow{\varphi^{\triangleleft n}} p^{\triangleleft n},$$

models n runs through the system φ.

Meanwhile, as the limit of the diagram (8.1) with the row of polynomials of the form $p^{\triangleleft n}$ across the top, the carrier $\mathfrak{t}_p$ of the cofree comonoid on p comes equipped with a lens $\epsilon_p^{(n)} \colon \mathfrak{t}_p \to p^{\triangleleft n}$ for each $n \in \mathbb{N}$. Since the retrofunctor $F \colon \mathcal{C} \nrightarrow \mathscr{T}_p$ has an underlying lens between carriers $f \coloneqq UF \colon \mathfrak{c} \to \mathfrak{t}_p$, we can obtain another lens $\mathfrak{c} \to^{\triangleleft n}$ as the composite

$$\mathfrak{c} \xrightarrow{f} \mathfrak{t}_p \xrightarrow{\epsilon_p^{(n)}} p^{\triangleleft n}.$$

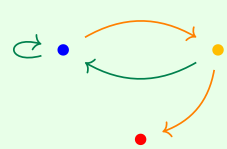
It follows from our forgetful-cofree adjunction that these two composites are equal.

Proposition 8.49. With the definitions above, the following diagram commutes:

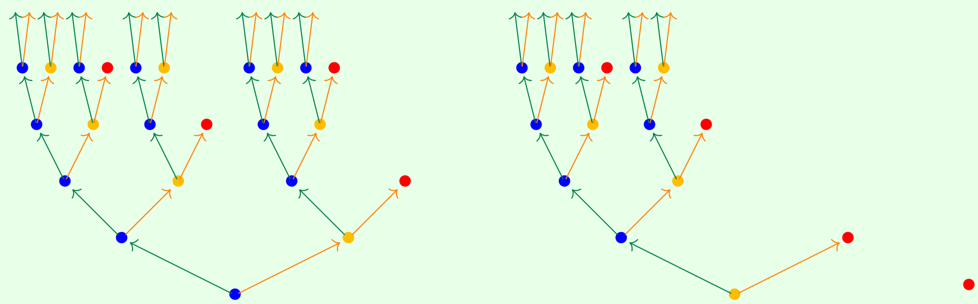
$$\begin{tikzcd}
\mathfrak{c} \arrow[r, "f"] \arrow[d, "\delta^{(n)}"'] & \mathfrak{t}_p \arrow[d, "\epsilon_p^{(n)}"] \\
\mathfrak{c}^{\triangleleft n} \arrow[r, "\varphi^{\triangleleft n}"'] & p^{\triangleleft n}.
\end{tikzcd}$$

Now we have a better sense of what we mean when we say that $F \colon \mathcal{C} \nrightarrow \mathscr{T}_p$ captures all the information that the lenses $\mathrm{Run}_n(\varphi) \colon \mathfrak{c} \to p^{\triangleleft n}$ encode: the category $\mathscr{T}_p$ is carried by a polynomial $\mathfrak{t}_p$ equipped with lenses $\epsilon_p^{(n)} \colon \mathfrak{t}_p \to p^{\triangleleft n}$, each of which exposes a part of the category as the n-fold interface $p^{\triangleleft n}$. All together, $\mathfrak{t}_p$ acts as a giant interface that captures the n-fold behavior of $p^{\triangleleft n}$ for every $n \in \mathbb{N}$. But to see all this explicitly, let's consider some examples.

Example 8.50. Let $S \coloneqq \{{\color{green}\bullet}, {\color{orange}\bullet}, {\color{red}\bullet}\}$ and $p \coloneqq y^2 + 1$, and consider the dynamical system $\varphi \colon Sy^S \to p$ modeling the halting deterministic state automaton from Exercise 4.23, depicted here again for your convenience:



Under the forgetful-cofree adjunction, the lens φ coincides with a retrofunctor $F \colon Sy^S \nrightarrow \mathscr{T}_p$ from the state category on S to the category of p-trees. The retrofunctor sends each state in $S = \{{\color{green}\bullet}, {\color{orange}\bullet}, {\color{red}\bullet}\}$ to a p-tree; these p-trees are drawn below (the first two are infinite). Then the vertices of each p-tree are sent back to a morphism in the state category; the color of each vertex indicates the codomain of the morphism to which that vertex is sent.



Each p-tree above, with its vertices colored, thus encodes all the ways to navigate the automaton. In particular, the *maximal rooted paths* of these trees (i.e. those that terminate at a leaf) trace out all the ways in which the automaton can halt, and therefore all the words that the automaton accepts. Notice, too, that every p-subtree of any one of these p-trees is another one of these three p-trees—do you see why?

In general, given a lens $\varphi\colon Sy^S \to y^A + 1$ modeling a halting deterministic state automaton and an initial state $s_0 \in S$, the $(y^A + 1)$-tree to which the corresponding retrofunctor $F\colon Sy^S \nrightarrow \mathscr{T}_{y^A+1}$ sends s_0 encodes the set of words accepted by the automaton with that initial state in its maximal rooted paths.

Example 8.51 (Languages recognized by deterministic state automata). Recall from Proposition 4.17 that a deterministic state automaton with a set of states S and a set of symbols A can be identified with a lens $y \to Sy^S$, indicating the initial state $s_0 \in S$, and a lens $\varphi\colon Sy^S \to 2y^A$, indicating the subset of accept states $F \coloneqq \varphi_1^{-1}(2) \subseteq S$ and the update function $u\colon S \times A \to S$ via $u(s, a) = \varphi_s^\sharp(a)$.

Under the forgetful-cofree adjunction, the lens φ coincides with a retrofunctor $F\colon Sy^S \nrightarrow \mathscr{T}_{2y^A}$. By Exercise 8.40, $\mathscr{T}_{2y^A}$ is the category of 2-labeled A-ary trees. So $F(s_0)$ is an element of $\mathsf{tree}_{2y^A} \cong 2^{\mathsf{List}(A)}$: an A-ary tree where each rooted path corresponds to a list of elements of A and bears one of the elements of 2, indicating whether the course through the automaton corresponding to that rooted path ends at an accept state or not. Equivalently, an element of $2^{\mathsf{List}(A)}$ is a subset of $\mathsf{List}(A)$; in the case of $F(s_0)$, it is the subset containing exactly the words in $\mathsf{List}(A)$ accepted by the automaton. So on objects, F sends each possible start state $s_0 \in S$ of the automaton to the subset of words that the automaton would then accept! Backward on morphisms, $F_{s_0}^\sharp$ sends every possible word to the state the automaton would reach if it followed that word starting from s_0.

Example 8.52 (Direction sequences to position sequences). We interpret our dynamical systems as converting sequences of directions to sequences of positions. The forgetful-cofree adjunction allows us to express this conversion formally in the language of **Poly**.

For convenience, we'll focus on the example of an (A, B)-Moore machine $\varphi\colon Sy^S \to By^A$, although of course we can generalize this beyond monomial interfaces.

The lens φ corresponds to a retrofunctor $F\colon Sy^S \nrightarrow \mathscr{T}_{By^A}$. By Exercise 8.40, $\mathscr{T}_{By^A}$ is the category of B-labeled A-ary trees; in paticular, its carrier is $\mathfrak{t}_{By^A} \cong B^{\mathsf{List}(A)} y^{\mathsf{List}(A)}$.

Then for every initial state $s_0 \in S$, the B-labeled A-ary tree $F(s_0)$ can be interpreted as a decision tree of all the possible sequences of directions that the system may receive. The label in B corresponding to the vertex (or rooted path) specified by each sequence in $\mathsf{List}(A)$ tells us the final position that the system returns when that sequence is fed in as directions. Put another way, $F(s_0) \in B^{\mathsf{List}(A)}$ can be interpreted as a function $\mathsf{List}(A) \to B$. So if the direction sequence is $(a_1, \ldots, a_n) \in A^n \subseteq \mathsf{List}(A)$, then the corresponding position sequence $(b_0, \ldots, b_n) \in B^{n+1}$ is given (non-recursively!) by

$$b_i \coloneqq F(s_0)(a_1, \ldots, a_i).$$

Finally, $F^\sharp_{s_0}(a_1, \ldots, a_i) \in S$ then corresponds to the system's state after that sequence of directions is given.

8.1.5 Morphisms between cofree comonoids

Given a lens $\varphi\colon p \to q$, the cofree functor gives us a comonoid morphism $\mathscr{T}_\varphi\colon \mathscr{T}_p \to \mathscr{T}_q$ as follows.

An object $t \in \mathsf{tree}_p$ is a tree; the tree $u \coloneqq \mathscr{T}_\varphi(t) \in \mathsf{tree}_q$ is constructed recursively as follows. If the root of t is $i \in p(1)$ then the root of u is $j \coloneqq \varphi_1(i)$. To each branch $b \in q[j]$, we need to assign a new tree, and we use the one situated at $\varphi^\sharp_i(b)$.

Exercise 8.53 (Solution here). Let $p \coloneqq \{\bullet\}y^2 + \{\bullet\}$ and $q \coloneqq \{\bullet, \bullet\}y + \{\bullet, \bullet\}$.
1. Choose a lens $\varphi\colon p \to q$, and write it out.
2. Choose a tree $T \in \mathsf{tree}_p$ with at least height 3.
3. What is $\mathscr{T}_\varphi(T)$? ◊

Exercise 8.54 (Solution here). Let p be a polynomial.
1. Show there is an induced retrofunctor $\mathscr{T}_p \to \mathscr{T}_{p^{\triangleleft n}}$ for all $n \in \mathbb{N}$.
2. When $n \geq 1$, is the induced retrofunctor is an isomorphism? ◊

8.1.6 Some categorical properties of cofree comonoids

Proposition 8.55. 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 counit lens $\pi\colon \mathsf{tree}_p \to p$ from Theorem 8.45 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 8.56. 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 8.55. □

Proposition 8.57. The additive monoid $y^{\mathbb{N}}$ of natural numbers has a $\times$-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 retrofunctors $e\colon y \to y^{\mathbb{N}}$ and $m\colon y^{\mathsf{List(2)}} \to y^{\mathbb{N}}$ that satisfy the axioms of a monoid.

The unique lens $y \to y^{\mathbb{N}}$ is a retrofunctor (it is the mate of the identity $y \to y$). We take m to be the mate of the lens $y^{\mathsf{List(2)}} \to y$ 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 7.71 that an arrow field of a category $\mathcal{C}$ is a retrofunctor $\mathcal{C} \nrightarrow y^{\mathbb{N}}$.

Corollary 8.58. For any category $\mathcal{C}$, the set $\mathbf{Cat}^\sharp(\mathcal{C}, y^{\mathbb{N}})$ of arrow fields 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 8.57. □

A retrofunctor $\mathcal{C} \nrightarrow y^{\mathbb{N}}$ 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 arrow fields.

Theorem 8.59. The arrow fields 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 7.79.

Proof. Let $\mathcal{C}$ be a category and $(M, e, *)$ a monoid. A retrofunctor $F\colon \mathcal{C} \nrightarrow y^M$ has no data on objects; it is just a way to assign to each $c \in \mathcal{C}$ and $m \in M$ a morphism $F^\sharp_c(m)\colon c \to c'$ for some $c' := \mathrm{cod}(F^\sharp_c(m))$. This assignment must send identities to identities and composites to composites: given $m' \in M$ we have $F^\sharp_c(m \mathbin{⨟} m') = F^\sharp_c(m) \mathbin{⨟} F^\sharp_{c'}(m')$. This is exactly the data of a monoid morphism $M \to \vec{\mathcal{C}}$: it assigns to each $m \in M$ an arrow field $\mathcal{C}$, preserving unit and multiplication. □

Proposition 8.60. There is a commutative square of left adjoints

$$\begin{array}{ccc} \mathbf{Mon}^{\mathrm{op}} & \xrightarrow{U} & \mathbf{Set}^{\mathrm{op}} \\ {\scriptstyle y^-}\big\downarrow & & \big\downarrow{\scriptstyle y^-} \\ \mathbf{Cat}^\sharp & \xrightarrow[U]{} & \mathbf{Poly} \end{array}$$

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 7.79, 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 5.12, that $U\colon \mathbf{Cat}^\sharp \to \mathbf{Poly}$ is a left adjoint in Theorem 8.45, and that $y^-\colon \mathbf{Mon} \to \mathbf{Cat}^\sharp$ is a left adjoint in Theorem 8.59. □

8.2 More categorical properties of Cat$^\sharp$

Many of the properties of **Poly** we covered in Chapters 3 and 5 have analogues in **Cat**$^\sharp$; we review these here.

8.2.1 Other special comonoids and adjunctions

We begin by highlighting a few other adjunctions involving **Cat**$^\sharp$, as well as the special comonoids in **Cat**$^\sharp$ they provide.

Proposition 8.61. The functor y^- from Proposition 8.60 factors through an isomorphism of categories

$$\mathbf{Cat}^\sharp_{\text{rep}} \cong \mathbf{Mon}^{\text{op}},$$

where **Mon** is the category of monoids and monoid homomorphisms and $\mathbf{Cat}^\sharp_{\text{rep}}$ is the full subcategory of $\mathbf{Cat}^\sharp$ consisting of categories with representable carriers y^M for some $M \in \mathbf{Set}$.

Proof. Let $\mathcal{C}$ be a category. It has only one object iff its carrier $\mathfrak{c}$ 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 retrofunctors between monoids are dual—opposite—to morphisms between monoids.

A retrofunctor $f : y^M \to y^N$ involves a single function $f^\sharp : N \to M$ that must satisfy a law coming from unitality and one coming from composition, as in Definition 7.49. The result can now be checked by hand, or seen formally as follows. Each object in the two diagrams of (7.49) is representable by Exercise 6.9. The Yoneda embedding $\mathbf{Set}^{\text{op}} \to \mathbf{Poly}$ is fully faithful, so these two diagrams are equivalent to the unit and composition diagrams for monoid homomorphisms. □

Exercise 8.62 (Solution here). Let $\mathbf{Cat}^\sharp_{\text{lin}}$ be the full subcategory of $\mathbf{Cat}^\sharp$ consisting of categories with linear carriers Sy for some $S \in \mathbf{Set}$. Show that there is an isomorphism of categories

$$\mathbf{Cat}^\sharp_{\text{lin}} \cong \mathbf{Set}. =$$

◊

Proposition 8.63 (Discrete categories)**.** The inclusion $\mathbf{Cat}^\sharp_{\text{lin}} \to \mathbf{Cat}^\sharp$ has a left adjoint sending each $(\mathfrak{c}, \epsilon, \delta) \in \mathbf{Cat}^\sharp$ to the unique comonoid carried by $(\mathfrak{c} \triangleleft 1)y$ in $\mathbf{Cat}^\sharp_{\text{lin}}$.

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 retrofunctor must pass identities back to identities. □

8.2.2 Vertical-cartesian factorization of retrofunctors

A retrofunctor is called *cartesian* if the underlying lens $f : \mathfrak{c} \to \mathfrak{d}$ is cartesian (i.e. for each position $i \in \mathfrak{c}(1)$, the map $f^\sharp_i : \mathfrak{d}[f_1(i)] \to \mathfrak{c}[i]$ is an isomorphism).

Proposition 8.64. Every retrofunctor $f\colon \mathcal{C} \nrightarrow \mathcal{D}$ factors as a vertical morphism followed by a cartesian morphism

$$\mathcal{C} \overset{\text{vert}}{\nrightarrow} \mathcal{C}' \overset{\text{cart}}{\nrightarrow} \mathcal{D}.$$

Proof. A retrofunctor $\mathcal{C} \nrightarrow \mathcal{D}$ is a lens $\mathfrak{c} \to \mathfrak{d}$ satisfying some properties, and any lens $f\colon \mathfrak{c} \to \mathfrak{d}$ 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 retrofunctors. 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 $\mathrm{cod}(m) := \mathrm{cod}(f_i^\sharp(m))$, and given an emanating morphism $m' \in \mathfrak{c}'[\mathrm{cod}(m)]$, assign the composite $m \mathbin{⨟} m'$ in $\mathfrak{c}'$ to be $m \mathbin{⨟} m'$ in $\mathfrak{d}$. In Exercise 8.65 we will check that with these definitions, $\mathfrak{c}'$ is a category and both g and h are retrofunctors. □

Exercise 8.65 (Solution here). We will complete the proof of Proposition 8.64, 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 8.66. The wide[a] subcategory of cartesian retrofunctors in **Cat**$^\sharp$ is isomorphic to the wide subcategory of discrete opfibrations in **Cat**.

[a] A subcategory **D** of a category **C** is *wide* if every object of **C** is in **D**.

Proof. Suppose that $\mathcal{C}$ and $\mathcal{D}$ are categories. Both a functor and a retrofunctor between them involve a map on objects, say $f\colon \mathrm{Ob}\,\mathcal{C} \to \mathrm{Ob}\,\mathcal{D}$. For any object $c \in \mathrm{Ob}\,\mathcal{C}$, a functor gives a function, say $f_\sharp\colon \mathcal{C}[c] \to \mathcal{D}[f(c)]$ whereas a retrofunctor gives a function $f^\sharp\colon \mathcal{D}[f(c)] \to \mathcal{C}[c]$. The retrofunctor 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 retrofunctor (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 8.67. The wide subcategory of vertical maps in $\mathbf{Cat}^\sharp$ is isomorphic to the opposite of the wide subcategory bijective-on-objects maps in $\mathbf{Cat}$:

$$\mathbf{Cat}^\sharp_{\text{vert}} \cong (\mathbf{Cat}_{\text{boo}})^{\text{op}}.$$

Proof. Let $\mathcal{C}$ and $\mathcal{D}$ be categories. Given a vertical retrofunctor $F\colon \mathcal{C} \nrightarrow \mathcal{D}$, 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 $G(f) \coloneqq F^\sharp_{G_1(d)}$. 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 retrofunctors.

The construction of a vertical retrofunctor from a bijective-on-objects functor is analogous, and it is easy to check that the two constructions are inverses. □

Exercise 8.68 (Solution here). Let S be a set and consider the state category $\mathcal{S} \coloneqq (Sy^S, \epsilon, \delta)$. Use Proposition 8.67 to show that categories $\mathcal{C}$ equipped with a vertical retrofunctor $\mathcal{S} \nrightarrow \mathcal{C}$ can be identified with categories whose set of objects has been identified with S. ◊

Exercise 8.69 (Solution here). Consider the categories $\mathcal{C} \coloneqq \boxed{\bullet \rightrightarrows \bullet}$ and $\mathcal{D} \coloneqq \boxed{\bullet \to \bullet}$. There is a unique bijective-on-objects (boo) functor $F\colon \mathcal{C} \to \mathcal{D}$ and two boo functors $G_1, G_2\colon \mathcal{D} \to \mathcal{C}$. These have corresponding retrofunctors going the other way.

1. Write down the morphism $\mathfrak{d} \to \mathfrak{c}$ of carriers corresponding to F.
2. Write down the morphism $\mathfrak{c} \to \mathfrak{d}$ of carriers corresponding to either G_1 or G_2. ◊

We record the following proposition here.

Proposition 8.70. If $\varphi\colon p \to q$ is a cartesian lens, then $\mathscr{T}_\varphi\colon \mathscr{T}_p \to \mathscr{T}_q$ is a cartesian retrofunctor: that is, for each $t \in \mathsf{tree}_p$, the on-morphisms function

$$(\mathscr{T}_\varphi)^\sharp_t : \mathscr{T}_q[\mathscr{T}_\varphi t] \xrightarrow{\cong} \mathscr{T}_p[t]$$

is a bijection.

Proof. Given $\varphi\colon p \to q$ cartesian, each tree $T \in \mathsf{tree}_p$ is sent to a tree in tree_q with the same branching profile. A morphism emanating from it is just a finite rooted path, and the set of these is completely determined by the branching profile. Thus we have the desired bijection. □

8.2.3 Limits and colimits of comonoids

We saw in Theorems 5.33 and 5.43 that **Poly** has all small limits and colimits. It turns out that $\mathbf{Cat}^\sharp$ has all small limits and colimits as well. We start by discussing colimits in $\mathbf{Cat}^\sharp$, as they are somewhat easier to get a handle on.

Colimits in $\mathbf{Cat}^\sharp$

It is a consequence of the forgetful-cofree adjunction from Theorem 8.45 that $\mathbf{Cat}^\sharp$ inherits all the colimits from **Poly**. As these results are somewhat technical, relying on a property of functors known as *comonadicity*, we defer their proofs to references.

Proposition 8.71 (Porst)**.** The forgetful functor $\mathbf{Cat}^\sharp \to \mathbf{Poly}$ is comonadic.

Proof. The fact that a forgetful functor $\mathbf{Cat}^\sharp \cong \mathbf{Comon}(\mathbf{Poly}) \to \mathbf{Poly}$ 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]. So the result follows from Theorem 8.45. □

Corollary 8.72. The category $\mathbf{Cat}^\sharp$ has all small colimits. They are created by the forgetful functor $\mathbf{Cat}^\sharp \to \mathbf{Poly}$.

Proof. A comonadic functor creates all colimits that exist in its codomain (see [nLa18]), and by Theorem 5.43, the category **Poly** has all small colimits. □

Example 8.73 (Coproducts in $\mathbf{Cat}^\sharp$). Probably the most familiar kind of colimit in $\mathbf{Cat}^\sharp$ is the coproduct, as Corollary 8.72 tells us that it agrees with the usual coproduct from **Cat**. Here's why.

For concreteness, let I be a set and $(\mathcal{C}_i)_{i\in I}$ be categories with carriers $(\mathfrak{c}_i)_{i\in I}$. Then the coproduct of $(\mathcal{C}_i)_{i\in I}$ in **Cat** is the category $\sum_{i\in I}\mathcal{C}_i$, whose objects are given by the disjoint union of the objects in each summand, so that

$$\operatorname{Ob}\left(\sum_{i\in I}\mathcal{C}_i\right)\cong\sum_{i\in I}\operatorname{Ob}\mathcal{C}_i=\sum_{i\in I}\mathfrak{c}_i(1)\cong\left(\sum_{i\in I}\mathfrak{c}_i\right)(1),$$

and whose morphisms out of each $c\in\operatorname{Ob}\mathcal{C}_j\subseteq\operatorname{Ob}\sum_{i\in I}\mathcal{C}_i$ are just the morphisms out of c in the summand $\mathcal{C}_j$, so

$$\left(\sum_{i\in I}\mathcal{C}_i\right)[c]\cong\mathcal{C}_j[c]=\mathfrak{c}_j[c]\cong\left(\sum_{i\in I}\mathfrak{c}_i\right)[c]$$

Hence $\sum_{i\in I}\mathcal{C}_i$ is carried by the coproduct of polynomials $\sum_{i\in I}\mathfrak{c}_i$.

It remains to show that $\sum_{i \in I} \mathcal{C}_i$ is also the coproduct of $(\mathcal{C}_i)_{i \in I}$ in $\mathbf{Cat}^\sharp$. We already know from Corollary 8.72 that it has the right carrier: the coproduct of carriers $\sum_{i \in I} \mathfrak{c}_i$. It also has the right morphisms, for any object (i, x) with $x \in \mathfrak{c}_i(1)$, the set of emanating morphisms is—and should be—$\mathfrak{c}_i[x]$.

Exercise 8.74 (Solution here).

1. Show that 0 has a unique comonoid structure.
2. Explain why 0 with its comonoid structure is initial in $\mathbf{Cat}^\sharp$ in two ways: by explicitly showing it has the required universal property, or by invoking Corollary 8.72. ◊

Exercise 8.75 (Solution here). Given a comonoid $(\mathfrak{c}, \epsilon, \delta) \in \mathbf{Cat}^\sharp$, show that there is an induced comonoid structure on the polynomial $2\mathfrak{c}$. ◊

Limits in $\mathbf{Cat}^\sharp$

As in the case of colimits, there is a rather technical result showing that the forgetful-cofree adjunction implies the existence of all small limits of comonoids, which we summarize here.

Corollary 8.76. The category $\mathbf{Cat}^\sharp$ has all small limits.

Proof. By Theorem 8.45, the forgetful functor $U\colon \mathbf{Cat}^\sharp \to \mathbf{Poly}$ has a right adjoint, and by Theorem 5.33, **Poly** itself has all small limits. Furthermore equalizers in **Poly** are connected limits, so by Theorem 6.80, they are preserved by $\triangleleft$ on either side. Then the result follows from [Por19, Fact 3.4]. □

8.2.4 Parallel product comonoids

The usual product of categories is not the categorical product in $\mathbf{Cat}^\sharp$. It is, however, a *monoidal* product on $\mathbf{Cat}^\sharp$, coinciding with the parallel product $\otimes$ on **Poly**.

Proposition 8.77. The parallel product $(y, \otimes)$ on **Poly** extends to a monoidal structure $(y, \otimes)$ on $\mathbf{Cat}^\sharp$, such that the forgetful functor $U\colon \mathbf{Cat}^\sharp \to \mathbf{Poly}$ is strong monoidal with respect to $\otimes$. The parallel product of two categories is their product in **Cat**.

Proof. Let $\mathcal{C}, \mathcal{D} \in \mathbf{Cat}^\sharp$ be categories corresponding to comonoids $(\mathfrak{c}, \epsilon_\mathfrak{c}, \delta_\mathfrak{c})$ and $(\mathfrak{d}, \epsilon_\mathfrak{d}, \delta_\mathfrak{d})$.

The carrier 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 $\delta_{\mathcal{C}\otimes\mathcal{D}}\colon (\mathfrak{c}\otimes\mathfrak{d})\to(\mathfrak{c}\otimes\mathfrak{d})\triangleleft(\mathfrak{c}\otimes\mathfrak{d})$ using the duoidal property:

$$\mathfrak{c}\otimes\mathfrak{d}\xrightarrow{\delta_{\mathfrak{c}}\otimes\delta_{\mathfrak{d}}}(\mathfrak{c}\triangleleft\mathfrak{c})\otimes(\mathfrak{d}\triangleleft\mathfrak{d})\to(\mathfrak{c}\otimes\mathfrak{d})\triangleleft(\mathfrak{c}\otimes\mathfrak{d}).$$

One can check that this says that codomains and composition are defined coordinate-wise, 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 8.78. □

Exercise 8.78 (Solution here). We complete the proof of Proposition 8.77.

1. Show that $(\mathfrak{c}\otimes\mathfrak{d},\epsilon_{\mathfrak{c}\otimes\mathfrak{d}},\delta_{\mathfrak{c}\otimes\mathfrak{d}})$, as described in Proposition 8.77, forms a comonoid.
2. Check that the construction $(\mathcal{C},\mathcal{D})\mapsto\mathcal{C}\otimes\mathcal{D}$ is functorial in $\mathcal{C},\mathcal{D}\in\mathbf{Cat}^\sharp$. ◊

The cofree construction works nicely with this monoidal product.

Proposition 8.79. The cofree functor $p\mapsto\mathscr{T}_p$ is lax monoidal; in particular there is a lens $y\to\mathfrak{t}_y$, and for any $p,q\in\mathbf{Poly}$ there is a natural lens

$$\mathfrak{t}_p\otimes\mathfrak{t}_q\to\mathfrak{t}_{p\otimes q}.$$

satisfying the usual conditions.

Proof. By Proposition 8.77, 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 oplax monoidal functor is lax monoidal. □

Exercise 8.80 (Solution here).

1. What polynomial is $\mathfrak{t}_y$?
2. What is the lens $y\to\mathfrak{t}_y$ from Proposition 8.79?
3. Explain in words how to think about the lens $\mathfrak{t}_p\otimes\mathfrak{t}_q\to\mathfrak{t}_{p\otimes q}$ from Proposition 8.79, for arbitrary $p,q\in\mathbf{Poly}$. ◊

8.3 Comodules over polynomial comonoids

Just as we can define a category of comonoids within any monoidal category, we can further define a notion of comodules over such comonoids. And much like how polynomial comonoids are categories, these comodules can also be described in categorical terms we are already familiar with. There is much more to say about comodules over polynomial comonoids than we have room for here—we will merely give a glimpse of the theory and applications.

8.3.1 Left and right comodules

When the monoidal category is not symmetric, left comodules and right comodules may differ significantly, so we define them separately (but notice that the diagrams for one are analogous to the diagrams for the other).

Definition 8.81 (Left comodule)**.** In a monoidal category $(\mathbf{C}, y, \triangleleft)$, let $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ be a comonoid. A *left $\mathscr{C}$-comodule* is

- an object $m \in \mathbf{C}$, called the *carrier*, equipped with
- a morphism $\mathfrak{c} \triangleleft m \xleftarrow{\lambda} m$ called the *left coaction*,

such that the following diagrams, collectively known as the *left comodule laws*, commute:

$$
\begin{tikzcd}
\mathfrak{c} \triangleleft m \arrow[d, "\epsilon \triangleleft m"'] & m \arrow[l, "\lambda"'] \arrow[dl, equal, bend left] \\
m &
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\mathfrak{c} \triangleleft m \arrow[d, "\delta \triangleleft m"'] & m \arrow[l, "\lambda"'] \arrow[d, "\lambda"] \\
\mathfrak{c} \triangleleft \mathfrak{c} \triangleleft m & \mathfrak{c} \triangleleft m \arrow[l, "\mathfrak{c} \triangleleft \lambda"]
\end{tikzcd}
\tag{8.82}
$$

When referring to a left $\mathscr{C}$-comodule, we may omit its coaction if it can be inferred from context (or simply unspecified), identifying the comodule with its carrier.

A *morphism* of left $\mathscr{C}$-comodules m and n is a morphism $\alpha \colon m \to n$ such that the following diagram commutes:

$$
\begin{tikzcd}
\mathfrak{c} \triangleleft m \arrow[d, "\mathfrak{c} \triangleleft \alpha"'] & m \arrow[l] \arrow[d, "\alpha"] \\
\mathfrak{c} \triangleleft n & n \arrow[l]
\end{tikzcd}
$$

Here the top and bottom morphisms are the left coactions of m and n.

Exercise 8.83 (Solution here). Show that the category of $\mathscr{C}$-coalgebras from Definition 7.96 is exactly the category of *constant* left $\mathscr{C}$-comodules—i.e. the full subcategory of the category of left $\mathscr{C}$-comodules spanned by those left $\mathscr{C}$-comodules whose carriers are constant polynomials. ◊

Definition 8.84 (Right comodule)**.** In a monoidal category $(\mathbf{C}, y, \triangleleft)$, let $\mathscr{D} = (\mathfrak{d}, \epsilon, \delta)$ be a comonoid. A *right $\mathscr{D}$-comodule* is

- an object $m \in \mathbf{C}$, called the *carrier*, equipped with
- a morphism $m \xrightarrow{\rho} m \triangleleft \mathfrak{d}$ called the *right coaction*,

such that the following diagrams, collectively known as the *right comodule laws*, commute:

$$
\begin{tikzcd}
m \arrow[r, "\rho"] \arrow[dr, equal, bend right] & m \triangleleft \mathfrak{d} \arrow[d, "m \triangleleft \epsilon"] \\
& m
\end{tikzcd}
\qquad
\begin{tikzcd}
m \arrow[r, "\rho"] \arrow[d, "\rho"'] & m \triangleleft \mathfrak{d} \arrow[d, "m \triangleleft \delta"] \\
m \triangleleft \mathfrak{d} \arrow[r, "\rho \triangleleft \mathfrak{d}"'] & m \triangleleft \mathfrak{d} \triangleleft \mathfrak{d}
\end{tikzcd}
\tag{8.85}
$$

When referring to a right $\mathcal{D}$-comodule, we may omit its coaction if it can be inferred from context (or unspecified), identifying the comodule with its carrier.

A *morphism* of right $\mathcal{D}$-comodules m and n is a morphism $\alpha\colon m \to n$ such that the following diagram commutes:

$$
\begin{tikzcd}
m \arrow[r] \arrow[d, "\alpha"'] & m \triangleleft \mathfrak{d} \arrow[d, "\alpha \triangleleft \mathfrak{d}"] \\
n \arrow[r] & n \triangleleft \mathfrak{d}
\end{tikzcd}
$$

Here the top and bottom morphisms are the right coactions of m and n.

Exercise 8.86 (Solution here).

1. Draw the equations of (8.82) using polyboxes.
2. Draw the equations of (8.85) using polyboxes. ◊

Exercise 8.87 (Solution here). Recall from Exercise 7.64 that y has a unique category structure.

1. Show that the category of left y-comodules is isomorphic to **Poly**.
2. Show that the category of right y-comodules is isomorphic to **Poly**. If it is similar, just say "similar"; if not, explain.

◊

We can characterize left comodules as something far more familiar.

Proposition 8.88. Let $\mathcal{C}$ be a category. The category of left $\mathcal{C}$-comodules is equivalent to the category of functors $\mathcal{C} \to$ **Poly**.

Sketch of proof. Let $\mathfrak{c}$ be the carrier of $\mathcal{C}$. Given a left $\mathcal{C}$-comodule $m \to \mathfrak{c} \triangleleft m$, we saw in Exercise 8.86 that there is an induced function $|-|\colon m(1) \to \mathfrak{c}(1)$, so to each object $i \in \mathrm{Ob}(\mathcal{C})$ we can associate a set $P_i^m := \{a \in m(1) \mid |a| = i\}$. We also have for each $a \in P_i \subseteq m(1)$ a set $m[a]$, and so we can consider the polynomial

$$p_i^m := \sum_{a \in P_i^m} y^{m[a]}.$$

The polynomial $p_i^m \in$ **Poly** is easily seen to be functorial in the comodule m. Moreover, given a morphism $f\colon i \to i'$ in $\mathcal{C}$, we have for each $a \in p_i^m(1)$ an element $a.f \in m(1)$, with $|a.f| = i'$, i.e. $a.f \in P^m_{i'} = p^m_{i'}(1)$. Similarly for each $x \in m[a.f] = p^m_{i'}[a.f]$ we have $\lambda^\sharp_a(f, x) \in m[a] = p^m_i[a]$, and thus we have a lens $p^m_i \to p^m_{i'}$. In this way we have obtained we obtain a functor $p^m_{-} \colon \mathcal{C} \to \mathbf{Poly}$, and this functor is itself functorial in m.

We leave the proof that this construction has an inverse to the reader. □

Similarly, a right comodule is not an altogether foreign concept.

Proposition 8.89. Let $\mathcal{D}$ be a category. A right $\mathcal{D}$-comodule m can be identified (up to isomorphism) with a functor $\mathcal{D} \to \mathbf{Set}^{m(1)}$.

Sketch of proof. Let $\mathfrak{d}$ be the carrier of $\mathcal{D}$. Given a right $\mathcal{D}$-comodule $m \to m \triangleleft \mathfrak{d}$ we need a functor $F \colon \mathcal{D} \times m(1) \to \mathbf{Set}$. We again rely heavily on Exercise 8.86. Given $j \in \mathcal{D}$ and $a \in m(1)$, define

$$F(j, d) \coloneqq \{x \in m[a] \mid |a.f| = j\}.$$

Given $g \colon j \to j'$ we get $\rho^\sharp_a(x, g) \in m[a]$ with $|a.\rho^\sharp_a(x, g)| = j'$, and thus g induces a function $F(g, d) \colon F(j, d) \to F(j', d)$. Again by laws from Exercise 8.86 these are functorial in g, completing the proof sketch. □

Proposition 8.90. Let $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ be a comonoid in $\mathbf{Poly}$. For any set G, the polynomial $y^G \triangleleft \mathfrak{c}$ has a natural right $\mathscr{C}$-comodule structure.

Proof. We use the lens $(y^G \triangleleft \delta) \colon (y^G \triangleleft \mathfrak{c}) \to (y^G \triangleleft \mathfrak{c} \triangleleft \mathfrak{c})$. 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 8.91. Functions $i' \colon G \to \mathfrak{c} \triangleleft 1$ are in bijection with positions $i \in y^G \triangleleft \mathfrak{c} \triangleleft 1$. Let $m \coloneqq i^*(y^G \triangleleft \mathfrak{c})$ and let ρ_i be the induced right $\mathscr{C}$-comodule structure. 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 $\mathscr{C}$-comodule, and we can identify it with a $\mathcal{C}$-set by Theorem 8.100. 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 comodule structure induced by Proposition 8.90 is indeed the free one. □

Exercise 8.92 (Solution here). Let $\mathcal{C}$ be a category and $i \in \mathcal{C}$ an object.

1. Consider i as a lens $y \to \mathfrak{c}$. Show that the vertical-cartesian factorization of this lens is $y \to y^{\mathfrak{c}[i]} \xrightarrow{\varphi} \mathfrak{c}$.
2. Use Proposition 6.88 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

$$\begin{tikzcd}
y^{\mathfrak{c}[i]} \arrow[r, "\delta^i"] \arrow[d, "\varphi"'] \arrow[dr, phantom, "\lrcorner", very near start] & y^{\mathfrak{c}[i]} \triangleleft \mathfrak{c} \arrow[d, "\text{cart}"] \\
\mathfrak{c} \arrow[r, "\delta"'] & \mathfrak{c} \triangleleft \mathfrak{c}
\end{tikzcd}$$

4. Show that this square is a pullback, as indicated.
5. Show that δ^i makes $y^{\mathfrak{c}[i]}$ a right $\mathscr{C}$-comodule. ◊

The lens 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 lens $y^{\mathfrak{c}[f]}\colon y^{\mathfrak{c}[i]} \to y^{\mathfrak{c}[i']}$.

Exercise 8.93 (Solution here).

1. Show that $y^{\mathfrak{c}[f]}$ is a right $\mathscr{C}$-comodules morphism.
2. Show that the construction $y^{\mathfrak{c}[f]}$ is functorial in f. ◊

Proposition 8.94. Let $\mathfrak{c}$ be a comonoid. For any set I and right $\mathfrak{c}$-comodules $(m_i)_{i\in I}$, the coproduct $m \coloneqq \sum_{i\in I} m_i$ has a natural right-comodule structure. Moreover, each representable summand in the carrier m of a right $\mathfrak{c}$-comodule is itself a right-$\mathfrak{c}$ comodule and m is their sum.

Sketch of proof. This follows from the fact in Proposition 6.47 that $- \triangleleft \mathfrak{c}$ commutes with coproduct, i.e. $(\sum_{i\in I} m_i) \triangleleft \mathfrak{c} \cong \sum_{i\in I}(m_i \triangleleft \mathfrak{c})$. □

Proposition 8.95. If $m \in \mathbf{Poly}$ is equipped with both a right $\mathscr{C}$-comodule and a right $\mathscr{D}$-comodule structure, we can naturally equip m with a $(\mathscr{C} \times \mathscr{D})$-comodule structure.

Proof. It suffices by Proposition 8.94 to assume that $m = y^M$ is representable. But a right $\mathscr{C}$-comodule with carrier y^M can be identified with a retrofunctor $My^M \to \mathscr{C}$.

Thus if y^M is both a right-$\mathscr{C}$ comodule and a right-$\mathscr{D}$ comodule, 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-$(\mathscr{C} \times \mathscr{D})$ comodule on y^M. □

8.3.2 Bicomodules

We take special note of any object that is both a left comodule and a right comodule in compatible ways.

Definition 8.96 (Bicomodule). In a monoidal category $(\mathbf{C}, y, \triangleleft)$, let $\mathcal{C}$ and $\mathcal{D}$ be comonoids with carriers $\mathfrak{c}$ and $\mathfrak{d}$. A $(\mathcal{C}, \mathcal{D})$-*bicomodule* is

1. an object $m \in \mathbf{C}$ that is both
2. a left $\mathcal{C}$-comodule, with left coaction $\mathfrak{c} \triangleleft m \xleftarrow{\lambda} m$, and
3. a right $\mathcal{D}$-comodule, with right coaction $m \xrightarrow{\rho} m \triangleleft \mathfrak{d}$,

such that the following diagram, known as the *coherence law*, commutes:

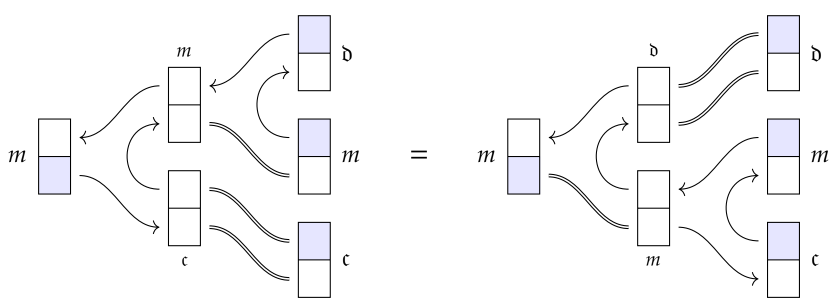
$$\begin{tikzcd}
& m \arrow[dl, "\lambda"'] \arrow[dr, "\rho"] & \\
\mathfrak{c} \triangleleft m \arrow[dr, "\mathfrak{c} \triangleleft \rho"'] & & m \triangleleft \mathfrak{d} \arrow[dl, "\lambda \triangleleft \mathfrak{d}"] \\
& \mathfrak{c} \triangleleft m \triangleleft \mathfrak{d}. &
\end{tikzcd} \tag{8.97}$$

We may denote such a bicomodule by $\mathcal{C} \overset{m}{\triangleleft\!\!\longleftarrow\!\!\triangleleft} \mathcal{D}$ when its left and right coactions have yet to be specified or may be inferred from context—or even by $\mathfrak{c} \overset{m}{\triangleleft\!\!\longleftarrow\!\!\triangleleft} \mathfrak{d}$ when the comonoid structures on $\mathfrak{c}$ and $\mathfrak{d}$ can be inferred from context as well.[a]

A *morphism* of $(\mathcal{C}, \mathcal{D})$-bicomodules is one that is a morphism of left $\mathcal{C}$-comodules and a morphism of right $\mathcal{D}$-comodules.

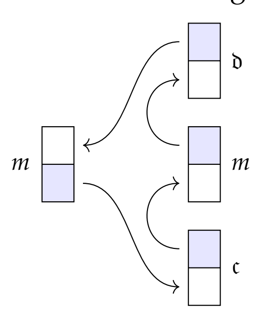
[a]The notation $\mathfrak{c} \triangleleft\!\!\longleftarrow\!\!\triangleleft \mathfrak{d}$ has a mnemonic advantage, as each $\triangleleft$ goes in the correct direction: $\mathfrak{c} \triangleleft m \leftarrow m$ and $m \to m \triangleleft \mathfrak{d}$. But it also looks like an arrow going backward from $\mathfrak{d}$ to $\mathfrak{c}$, which will turn out to have a semantic advantage as well.

We draw the commutativity of (8.97) using polyboxes.



(8.98)

This polybox equation implies that we can unambiguously write



for any bicomodule $\mathfrak{c} \xleftarrow{m} \mathfrak{d}$.

Exercise 8.99 (Solution here). Let $\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$ be a category. Recall from Exercise 7.64 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 $\mathbf{Poly} \cong {}_y\mathbf{Mod}_y$. ◊

8.3.3 More equivalences

We have seen that a retrofunctor from a state category to a category $\mathcal{C}$ carries the same data as a $\mathcal{C}$-coalgebra, which is in turn equivalent to a discrete opfibration over $\mathcal{C}$ or a copresheaf on $\mathcal{C}$. We then showed that cartesian retrofunctors to $\mathcal{C}$ is an equivalent notion as well. We are now ready to show that several kinds of comodules are also equivalent to all of these concepts.

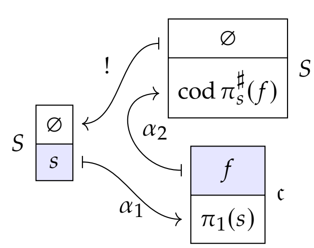
Theorem 8.100. Given a polynomial comonoid $\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$, the following comprise equivalent categories:

1. functors $\mathcal{C} \to \mathbf{Set}$;
2. discrete opfibrations over $\mathcal{C}$;
3. cartesian retrofunctors to $\mathscr{C}$;
4. $\mathscr{C}$-coalgebras (sets with a coaction by $\mathscr{C}$);
5. constant left $\mathscr{C}$-comodules;
6. $(\mathscr{C}, 0)$-bicomodules;
7. linear left $\mathscr{C}$-comodules; and
8. representable right $\mathscr{C}$-comodules (opposite).

In fact, all but the first comprise isomorphic categories; and up to isomorphism, any one of these can be identified with a retrofunctor from a state category to $\mathcal{C}$.

Proof. $1 \simeq 2 \cong 3$**:** This was shown in Proposition 7.108.

$3 \cong 4$**:** Given a cartesian retrofunctor $(\pi_1, \pi^\sharp)\colon \mathcal{S} \nrightarrow \mathcal{C}$, let $S \coloneqq \mathrm{Ob}(\mathcal{S})$ 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



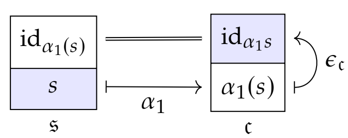
We check that this is indeed a coalgebra using properties of retrofunctors. For identities in $\mathscr{C}$, we have

$$\begin{aligned}\alpha_2(s, \mathrm{id}_{\pi_1(s)}) &= \operatorname{cod} \pi_s^\sharp(\mathrm{id}_{\pi_1(s)}) \\ &= \operatorname{cod} \mathrm{id}_s = s\end{aligned}$$

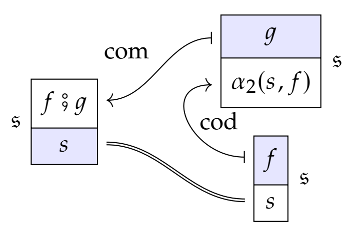
and for compositions in $\mathscr{C}$ we have

$$\begin{aligned}\alpha_2(s, f \mathbin{⨾} g) &= \operatorname{cod}\left(\pi_s^\sharp(f \mathbin{⨾} g)\right) \\ &= \operatorname{cod}\left(\pi_s^\sharp(f) \mathbin{⨾} \pi^\sharp_{\operatorname{cod}\pi_s^\sharp(f)} g\right) \\ &= \operatorname{cod}\left(\pi^\sharp_{\operatorname{cod}\pi_s^\sharp(f)} g\right) \\ &= \alpha_2(\alpha_2(s, f), g).\end{aligned}$$

Going backward, 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 lens $\varphi := (\alpha_1, \mathrm{id}) \colon \mathfrak{s} \to \mathfrak{c}$ to be the base change from Proposition 5.72. We need to show $\mathfrak{s}$ has a comonoid structure $(\mathfrak{s}, \epsilon, \delta)$ and that φ is a retrofunctor. We simply define the eraser $\epsilon \colon \mathfrak{s} \to y$ using α_1 and the eraser on $\mathfrak{c}$:



We give the duplicator $\delta \colon \mathfrak{s} \to \mathfrak{s} \triangleleft \mathfrak{s}$ using α_2 for the codomain, and using the composite $⨾$ from $\mathfrak{c}$:



In Exercise 8.101 we check that $(\mathfrak{s}, \epsilon, \delta)$ really is a comonoid, that $(\alpha_1, \mathrm{id}) \colon \mathfrak{s} \nrightarrow \mathfrak{c}$ is a retrofunctor, that the roundtrips between cartesian retrofunctors and coalgebras are identities, and that these assignments are functorial.

$4 \cong 5$: This is straightforward and was mentioned in Definition 7.96.

$5 \cong 6$: A right 0-comodule is in particular a polynomial $m \in \mathbf{Poly}$ and a lens $\rho \colon m \to m \triangleleft 0$ such that $(m \triangleleft \epsilon) \circ \rho = \mathrm{id}_m$. This implies ρ is monic, which itself implies by Proposition 5.18 that m must be constant since $m \triangleleft 0$ 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 (8.85) commute.

5 ≅ 7: By the adjunction in Theorem 5.4 and the fully faithful inclusion **Set** → **Poly** of sets as constant polynomials, Proposition 5.2, we have isomorphisms

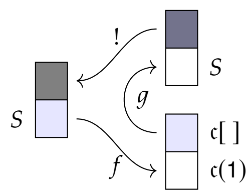
$$\mathbf{Poly}(Sy, \mathfrak{c} \triangleleft Sy) \cong \mathbf{Set}(S, \mathfrak{c} \triangleleft Sy \triangleleft 1) = \mathbf{Set}(S, \mathfrak{c} \triangleleft S) \cong \mathbf{Poly}(S, \mathfrak{c} \triangleleft S).$$

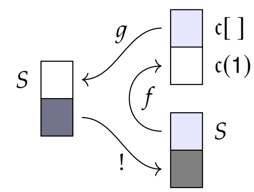
One checks easily that if $Sy \to \mathfrak{c} \triangleleft Sy$ corresponds to $S \to \mathfrak{c} \triangleleft S$ under the above isomorphism, then one is a left comodule if and only if the other is.

7 ≅ 8: By (6.66) 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 7.109. □

Exercise 8.101 (Solution here). Complete the proof of Theorem 8.100 (3 ≅ 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 retrofunctor.
3. Show that the roundtrips between cartesian retrofunctors and coalgebras are identities.
4. Show that the assignment of a $\mathscr{C}$-coalgebra to a cartesian retrofunctor over $\mathcal{C}$ is functorial.
5. Show that the assignment of a cartesian retrofunctor over $\mathcal{C}$ to a $\mathscr{C}$-coalgebra is functorial. ◊

Let $\mathcal{C}$ be a category. Under the above correspondence, the terminal functor $\mathcal{C} \to$ **Set** corresponds to the identity discrete opfibration $\mathcal{C} \to \mathcal{C}$, the identity retrofunctor $\mathscr{C} \to \mathscr{C}$, a certain left $\mathscr{C}$ comodule with carrier $\mathscr{C}(1)y$ which we call the *canonical left $\mathscr{C}$-comodule*, a certain constant left $\mathscr{C}$ comodule with carrier $\mathscr{C}(1)$ which we call the *canonical $(\mathscr{C}, 0)$-bicomodule*, and a certain representable right $\mathscr{C}$-comodule with carrier $y^{\mathscr{C}(1)}$ which we call the *canonical right $\mathcal{C}$-comodule*.

Exercise 8.102 (Solution here). For any object $c \in \mathcal{C}$, consider the representable functor $\mathcal{C}(c, -)\colon \mathcal{C} \to$ **Set**. What does it correspond to as a

1. discrete opfibration over $\mathcal{C}$?
2. cartesian retrofunctor to $\mathscr{C}$?
3. linear left $\mathscr{C}$-comodule?

4. constant left $\mathscr{C}$-comodule?
5. $(\mathscr{C}, 0)$-bicomodule?
6. representable right $\mathscr{C}$-comodule?
7. dynamical system with comonoid interface $\mathscr{C}$? ◊

Exercise 8.103 (Solution here). We saw in Theorem 8.100 that the category ${}_{\mathcal{C}}\mathbf{Mod}_0$ of $(\mathcal{C}, 0)$-bicomodule has a very nice structure: it's the topos of copresheaves on $\mathcal{C}$.

1. What is a $(0, \mathcal{C})$-bicomodule?
2. What is ${}_0\mathbf{Mod}_{\mathcal{C}}$? ◊

8.3.4 Bicomodules are parametric right adjoints

There is an equivalent characterization of bicomodules between a pair of polynomial comonoids due to Richard Garner. Thus we attribute the foundational theory of $\mathbf{Cat}^\sharp$ to Ahman-Uustalu-Garner.

Proposition 8.104 (Garner)**.** Let $\mathcal{C}$ and $\mathcal{D}$ be categories. Then the following can be identified, up to isomorphism:

1. a $(\mathcal{C}, \mathcal{D})$-bicomodule.
2. a parametric right adjoint $\mathbf{Set}^{\mathcal{D}} \to \mathbf{Set}^{\mathcal{C}}$.
3. a connected limit-preserving functor $\mathbf{Set}^{\mathcal{D}} \to \mathbf{Set}^{\mathcal{C}}$.

Parametric right adjoints model *data migrations* between categorical databases; see [SW15].

When a polynomial

$$m := \sum_{i \in m(1)} y^{m[i]}$$

is given the structure of a $(\mathcal{D}, \mathcal{C})$-bicomodule, the symbols in that formula are given a hidden special meaning:

$$m(1) \in \mathbf{Set}^{\mathcal{D}} \qquad \text{and} \qquad m[i] \in \mathbf{Set}^{\mathcal{C}}$$

Before we knew about bicomodule structures, what we called positions and directions were understood as each forming an ordinary set. In the presence of a bicomodule 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.

Definition 8.105 (Prafunctor)**.** Let $\mathcal{C}$ and $\mathcal{D}$ be categories. A *prafunctor* (also called a *parametric right adjoint functor*) $\mathbf{Set}^{\mathcal{C}} \to \mathbf{Set}^{\mathcal{D}}$ is one satisfying any of the conditions of Proposition 8.104.

8.3.5 Bicomodules in dynamics

We conclude with a peek at how bicomodules can model dynamical systems themselves.

Example 8.106 (Cellular automata). In Example 4.66 and Exercise 4.67 we briefly discussed cellular automata; here we will discuss another way that cellular automata show up, this time in terms of bicomodules.

Suppose that src, tgt: $A \rightrightarrows V$ 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 bicomodule structure

$$Vy \xleftarrow{g} Vy$$

where the right structure lens uses tgt; see Exercise 8.107 for details. A bicomodule

$$Vy \xleftarrow{T} 0$$

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

$$\begin{tikzcd} Vy & Vy \arrow[l, "g"'] \arrow[l, phantom, shift right=4, ""] & 0 \arrow[l, "T"'] \arrow[ll, bend left, "T"] \arrow[l, phantom, shift left=4, "\Downarrow\alpha"] \end{tikzcd}$$

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 $Vy \xleftarrow{g} Vy \xleftarrow{V} 0$ is again V.

Exercise 8.107 (Solution here). Let src, tgt: $A \rightrightarrows V$ and g and T be as in Example 8.106.

1. Give the structure lens $\lambda\colon g \to Vy \triangleleft g$
2. Give the structure lens $\rho\colon g \to g \triangleleft Vy$.

3. Give the set T and the structure lens $T \to Vy \triangleleft T$ corresponding to the functor $V \to \mathbf{Set}$ that assigns 2 to each vertex. ◊

Example 8.108 (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 8.106, this is all given by a diagram

$$\begin{tikzcd}
Vy & Vy \arrow[l, "g"'] \arrow[d, phantom, "\Downarrow\alpha"] & 0 \arrow[l, "T"'] \arrow[ll, bend left, "T"] \\
 & {} &
\end{tikzcd}$$

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 bicomodules

$$\begin{tikzcd}
Vy & 0 \arrow[l, bend right, "V"'] \arrow[l, bend left, "T"] \arrow[l, phantom, "\Downarrow\sigma"]
\end{tikzcd} \tag{8.109}$$

Note that g is a profunctor, i.e. for each $v \in V$ the summand $g_v = y^{\mathrm{src}^{-1}(v)}$ is representable, so it preserves the terminal object. In other words $g \triangleleft_{Vy} V \cong V$.

Now to run the cellular automaton on that initialization for $k \in \mathbb{N}$ steps is given by the composite

$$\begin{tikzcd}
Vy \arrow[d, equal] & \cdots \arrow[l, "g"'] & Vy \arrow[l, "g"'] \arrow[d, equal] & Vy \arrow[l, "g"'] \arrow[d, equal] & 0 \arrow[l, "V"'] \arrow[d, equal] \arrow[llll, bend right, "V"'] \arrow[d, phantom, shift right=4ex, "\Downarrow\sigma"] \\
Vy \arrow[d, equal] & \cdots \arrow[l, "g"'] & Vy \arrow[l, "g"'] \arrow[d, equal] & Vy \arrow[l, "g"'] \arrow[d, phantom, "\Downarrow\alpha"] & 0 \arrow[l, "T"'] \arrow[d, equal] \\
Vy \arrow[d, equal] & \cdots \arrow[l, "g"'] & Vy \arrow[l, "g"'] \arrow[d, Rightarrow, "\alpha\circ\cdots\circ\alpha"] & & 0 \arrow[ll, "T"'] \arrow[d, equal] \\
Vy & & & & 0 \arrow[llll, "T"]
\end{tikzcd}$$

Exercise 8.110 (Solution here). Explain why (8.109) models an initialization, i.e. a way to choose a starting color in each vertex. ◊

8.4 Summary and further reading

In this chapter we gave more of the theory of $\triangleleft$-comonoids, sometimes called polynomial comonads. We began by defining an adjunction

$$\begin{tikzcd}\mathbf{Cat}^\sharp \arrow[r, shift left, "U"] \arrow[r, phantom, "\Rightarrow"] & \mathbf{Poly} \arrow[l, shift left, "\mathscr{T}_-"]\end{tikzcd}$$

where $\mathscr{T}_p$ is the cofree comonoid $\mathscr{T}_p$ associated to any polynomial p, and we gave intuition for it in terms of p-trees. A p-tree is a (possibly infinite) tree for which each vertex is labeled by a position $i \in p(1)$ and its outgoing arrows are each labeled by an element of $p[i]$. The category corresponding to $\mathscr{T}_p$ has p-trees as its objects; the morphisms emanating from such a p-tree are the finite rooted paths up the tree, and the codomain of such a path is the tree rooted at its endpoint.

We then briefly discussed some other properties of $\mathbf{Cat}^\sharp$ including a formal proof of the fact that it has all limits and colimits. Colimits in $\mathbf{Cat}^\sharp$ are created by, i.e. fit nicely with, colimits in **Poly**, but limits are quite strange; we did not really discuss them here, but for example the product in $\mathbf{Cat}^\sharp$ of the walking arrow $\boxed{\bullet \to \bullet}$ with itself has infinitely-many objects!

We then moved on to left-, right-, and bicomodules between polynomial comonoids. In particular, we showed that left $\mathcal{C}$-comodules can be identified with functors $\mathcal{C} \to \mathbf{Poly}$, and that $(\mathcal{C}, \mathcal{D})$-bicomodules correspond to parametric right adjoint functors (prafunctors) $\mathbf{Set}^{\mathcal{D}} \to \mathbf{Set}^{\mathcal{C}}$. This idea was due to Richard Garner; it is currently unpublished, but can be found in video form here: `https://www.youtube.com/watch?v=tW6HYnqn6eI`. What we call prafunctors are sometimes called familial functors between (co-)presheaf categories; see [Web07], [GH18], and [Sha21] for more on this notion.

8.5 Exercise solutions

Solution to Exercise 8.5.

With $q \coloneqq y^2 + 3y^1$, every vertex of a q-tree starting from the root has either 1 or 2 outgoing edges, from which it follows that the tree is infinite and that every vertex has infinitely many descendents.

Solution to Exercise 8.11.

1. To select a $\mathsf{1}$-tree, we must select a position from $\mathsf{1}$... and then we are done, because there are no directions. So there is a unique $\mathsf{1}$-tree given by a root and no edges, implying that $\mathsf{tree}_\mathsf{1} \cong \mathsf{1}$.
2. To select a $\mathsf{2}$-tree, we must select a position from $\mathsf{2}$. Then we are done, because there are no directions. So every $\mathsf{2}$-tree is a root with no edges, and the root is labeled with one of the elements of $\mathsf{2}$. Hence $\mathsf{tree}_\mathsf{2} \cong \mathsf{2}$.
3. To select a y-tree, we must select a position from $\mathsf{1}$; then for the unique direction at that position, we must select a position from $\mathsf{1}$; and so forth. Since we only ever have 1 position to choose from, there is only 1 such y-tree we can build: an infinite ray extending from the root, where every vertex has exactly 1 child. Hence $\mathsf{tree}_y \cong \mathsf{1}$.
4. To select a y^2-tree, we must select a position from $\mathsf{1}$; then for each direction at that position, we must select a position from $\mathsf{1}$; and so forth. We only ever have 1 position to choose from, so there

is only 1 such y^2-tree: an infinite binary tree, where every vertex has exactly 2 children. Hence $\mathsf{tree}_{y^2} \cong 1$.

5. To select a $2y$-tree, we must select a position from 2; then for the unique direction at that position, we must select a position from 2; and so forth, one position out of 2 for each level of the tree. So every $2y$-tree is an infinite ray, where every vertex is labeled with an element of 2 and has exactly 1 child. There is then a unique vertex of height n for each $n \in \mathbb{N}$, so there is a bijection between $2y$-trees T and functions $f : \mathbb{N} \to 2$, where $f(n) \in 2$ is the label of the height-n vertex of T. Hence $\mathit{tree}_{2y} \cong 2^{\mathbb{N}}$.
6. To select a $(y+1)$-tree, we choose either a position with 1 direction or a position with no directions: if the position has no directions, we stop, and if the position has 1 direction, we repeat our choice for the next level. So a choice of $(y + 1)$-tree is equivalent to a choice of when, if ever, to stop. That is, for each $n \in \mathbb{N}$, there is a unique $(y+1)$-tree of height-n consisting of $n+1$ vertices along a single path, so that every vertex aside from the height-n leaf has exactly 1 child. Then there is exactly one more $(y + 1)$-tree: an infinite ray, obtained by always picking the position with 1 directions and never the one with no directions. Hence we could write $\mathsf{tree}_{y+1} \cong \mathbb{N} \cup \{\infty\}$ (although this set is in bijection with $\mathbb{N}$).
7. To select a By^A-tree for fixed $A, B \in \mathbf{Set}$, we must select a position from B; then for each direction in A at that position, we must select a position from B; and so forth. This yields a tree in which every vertex has A-many children and a label from B. Such a tree has 1 root and $|A|$ times as many vertices in one level than the level below it, so its height-n vertices for each $n \in \mathbb{N}$ are in bijection with the set A^{n}: each n-tuple in A^{n} gives a length-n rooted path of directions in A to follow up the tree, uniquely specifying a height-n vertex. So the set of all vertices is given by $\sum_{n\in\mathbb{N}} A^{\mathsf{n}} = \mathsf{List}(A)$. Then specifying a By^A-tree amounts to assigning a label from B to every vertex via a function $\mathsf{List}(A) \to B$. Hence $\mathsf{tree}_{By^A} \cong B^{\mathsf{List}(A)}$.

Solution to Exercise 8.12.

1. Yes: by Exercise 8.11 #11 in the case of $B = 1$, or by analogy with Exercise 8.11 #7 and #8, we have that $\mathsf{tree}_{y^{\mathsf{n}}}$ is the set of n-ary trees.
2. Yes: by Exercise 8.11 #11, we have that $\mathsf{tree}_{Ly^{\mathsf{n}}}$ is the set of L-labeled n-ary trees.

Solution to Exercise 8.14.

We defined tree_p in Definition 8.2 as a rooted tree T where every vertex v is labeled with an element of $p(1)$, together with a choice of bijection between the set of v's children and the set $p[i]$. We need to translate between that description and the description as the set-theoretic limit

$$1 \leftarrow p(1) \leftarrow p(p(1)) \leftarrow \cdots$$

An element of this set consists of an element of 1, which is determined, an element $i \in p(1)$, an element $(i_1, i_2) \in \sum_{i\in p(1)} \prod_{d\in p[i]} p(1)$ that "agrees" with i, an element of $\sum_{i_1\in p(1)} \prod_{d_1\in p[i_1]} \sum_{i_2\in p(1)} \prod_{d_2\in p[i_2]} p(1)$ that agrees, etc. But all this agreement is easy to simplify: we need elements

$$\begin{aligned}
&i_1 \in p(1),\\
&i_2 \in \prod_{d_1\in p[i_1]} p(1),\\
&i_3 \in \prod_{d_1\in p[i_1]} \prod_{d_2\in p[i_2 d_1]} p(1),\\
&i_4 \in \prod_{d_1\in p[i_1]} \prod_{d_2\in p[i_2 d_1]} \prod_{d_3\in p[i_3 d_1 d_2]} p(1)\\
&\vdots
\end{aligned}$$

To go from the former description to the latter, take such a rooted tree T and let i_1 be the label on the root vertex. Since by assumption there is a bijection between the set children of that vertex and the set

$p[i_1]$, take $i_2 \colon p[i_1] \to p(1)$ to assign to each $d_1 \in p[i_1]$ the label on the associated child vertex. Repeat this indefinitely.

To go from the latter description to the former, simply invert the description in the previous paragraph.

Solution to Exercise 8.16.

1. To give a function $t \colon \mathsf{tree}_p \to p(\mathsf{tree}_p)$ we need to choose, for each tree $T \in \mathsf{tree}_p$, an element $i \in p(1)$ and a function $T.- \colon p[i] \to \mathsf{tree}_p$. Choose i to be the label on the root of T, and choose $T.d$ to be the tree obtained by following the branch labeled d for any $d \in p[i]$.
2. Given an arbitrary coalgebra (S, f), where $S \in \mathbf{Set}$ and $f \colon S \to p \triangleleft S$, we need to show that there is a unique function $u \colon S \to \mathsf{tree}_p$ that commutes with f and t, i.e. with $t \circ u = p(f)$.
   For each n, we have a function $f_n \colon S \to p^{\triangleleft n} \triangleleft S$ by induction: for $n = 0$ use the identity function $S \to S$ and given $S \to p^{\triangleleft n} \triangleleft S$ we compose with $p^{\triangleleft n} \triangleleft f$ to get $S \to p^{\triangleleft n} \triangleleft p \triangleleft S$. Let $f'_n \colon S \to p^{\triangleleft n}(1)$ be given by $f'_n \coloneqq f_n \mathbin{⨾} (p^{\triangleleft n} \triangleleft !$. We know by Exercise 8.14 that tree_p is the limit of $p^{\triangleleft n}(1)$, so we get a unique map $(f'_n)_{n \in \mathbb{N}} \colon S \to \mathsf{tree}_p$. It commutes with f and t by construction, and the fact that it is constructed using the universal property of limits implies that it is appropriately unique.
3. The function $\mathsf{tree}_p \to p \triangleleft \mathsf{tree}_p$ from the first part is invertible. Indeed, given $i \in p(1)$ and $T' \colon p[i] \to \mathsf{tree}_p$, we construct an element of tree_p by taking its root to be labeled with i, its set of children to be $p[i]$, and for each $d \in p[i]$ taking the remaining tree to be $T'(d)$.

Solution to Exercise 8.17.

For the given values of p, we have already characterized the position-set $\mathsf{t}_p(1) \cong \mathsf{tree}_p$ of t in Exercise 8.11, so it remains to characterize the directions at each position, i.e. the vertices of each p-tree.

1. We saw that $\mathsf{tree}_1 \cong 1$ and that the unique 1-tree has a single vertex, its root. Equivalently, it has only 1 rooted path: the empty path from the root to itself. So $\mathsf{t}_1 \cong 1y^1 \cong y$.
2. We saw that $\mathsf{tree}_2 \cong 2$ and that each 2-tree has a single vertex, its root. Equivalently, its only rooted path is the empty path. So $\mathsf{t}_2 \cong 2y^1 \cong 2y$.
3. We saw that $\mathsf{tree}_y \cong 1$ and that the unique y-tree is a single ray extending from the root, where every vertex has exactly 1 child. So there is exactly 1 vertex of height-n for each $n \in \mathbb{N}$, yielding a bijection between the vertices of the y-tree to $\mathbb{N}$. Equivalently, the ray has exactly 1 rooted path of length n for each $n \in \mathbb{N}$. So $\mathsf{t}_y \cong y^{\mathbb{N}}$.
4. We saw that $\mathsf{tree}_{y^2} \cong 1$ and that the unique y^2-tree is an infinite binary tree, where every vertex has exactly 2 children. So the vertices of height-n are in bijection with 2^{n}, yielding a bijection between the vertices of the y^2-tree to $\sum_{n \in \mathbb{N}} 2^{\mathsf{n}} \cong \mathsf{List}(2)$. Equivalently, the rooted paths of an infinite binary tree are just finite binary sequences, which comprise the set $\mathsf{List}(2)$. Hence $\mathsf{t}_{y^2} \cong y^{\mathsf{List}(2)}$.
5. We saw that $\mathit{tree}_{2y} \cong 2^{\mathbb{N}}$, and that every $2y$-tree is an infinite ray, whose vertices (or rooted paths) are in bijection with $\mathbb{N}$. Hence $\mathsf{t}_{2y} \cong 2^{\mathbb{N}} y^{\mathbb{N}}$.
6. We saw that $\mathsf{tree}_{y+1} \cong \mathbb{N} \cup \{\infty\}$. Here the $(y+1)$-tree corresponding to $n \in \mathbb{N}$ consists of $n+1$ vertices along a single path, so that every vertex aside from the height-n leaf has exactly 1 child; and the $(y+1)$-tree corresponding to ∞ is an infinite ray. Thus the direction-set of t_{y+1} at $n \in \mathbb{N}$ is $\mathsf{n}+1$, while its direction-set at ∞ is $\mathbb{N}$. Hence $\mathsf{t}_{y+1} \cong \{\infty\} y^{\mathbb{N}} + \sum_{n \in \mathbb{N}} y^{\mathsf{n}+1}$.
7. We saw that $\mathsf{tree}_{By^A} \cong B^{\mathsf{List}(A)}$, where the height-$n$ vertices of a given By^A-tree are in bijection with the set A^{n}, so the set of all vertices of a By^A-tree is given by $\sum_{n \in \mathbb{N}} A^{\mathsf{n}} = \mathsf{List}(A)$. Equivalently, the rooted paths of the By^A-tree are just finite sequences in A, which comprise the set $\mathsf{List}(A)$. Hence $\mathsf{t}_{By^A} \cong B^{\mathsf{List}(A)} y^{\mathsf{List}(A)}$.

Solution to Exercise 8.36.

1. We compute t_B as follows. To select a B-tree, we must select a position from B. Then we are done, because there are no directions. So every B-tree consists of 1 vertex labeled with an element of B. Hence the set of B-trees $\mathsf{t}_B(1) = \mathsf{tree}_B$ is in bijection with B itself, yielding $\mathsf{t}_B \cong By$. (We could have also applied Exercise 8.17 #7 in the case of $A = 0$, yielding $\mathsf{t}_{By^0} \cong B^{\mathcal{List}(0)} y^{\mathsf{List}(0)} \cong By$, as $\mathsf{List}(0) \cong 1 + 0^1 + 0^2 + \cdots \cong 1$—the empty sequence is the only sequence with elements in 0.)

2. We characterize the B-tree category $\mathscr{T}_B$ as follows. We saw in Exercise 7.27 and Exercise 7.36 that (up to isomorphism) there is a unique comonoid structure on By, corresponding to the discrete category on B. So $\mathscr{T}_B$ must be isomorphic to the discrete category on B. In the language of B-trees, the objects of $\mathscr{T}_B$ are trees with exactly 1 vertex, labeled with an element of B. Each tree has an identity morphism, its root; but there are no vertices aside from the root, so there are no morphisms aside from the identities. In particular, the only B-subtree of any one B-tree is the entire B-tree itself, so there are no morphisms from one B-tree to a different B-tree. So the category of B-trees can indeed be identified with the discrete category on B.

Solution to Exercise 8.40.

We characterize $\mathscr{T}_{By^A}$, following Example 8.39.

- The objects are By^A-trees. By Exercise 8.17 #7, or Exercise 8.12 #2 with $L := B$ and n replaced with an arbitrary set A, a By^A-tree is a (infinite, assuming $|A|, |B| > 0$) tree whose every vertex is labeled by an element of B and whose children are in bijection with A. We can call these B-labeled A-ary trees; they are in bijection with the set $B^{\mathsf{List}(A)}$.
- A morphism out of a B-labeled A-ary tree is a finite list of directions in A, or a rooted path in an A-ary tree. They comprise the set $\mathsf{List}(A)$. The codomain of each rooted path is the B-labeled A-ary subtree rooted at the end of the path.
- The identity morphism on a given B-labeled A-ary tree is its empty rooted path.
- The composite of two lists in A is obtained by concatenation.

Solution to Exercise 8.41.

We characterize $\mathscr{T}_{y+1}$ as follows.

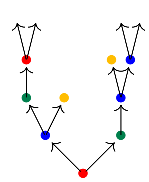
- The objects are $(y + 1)$-trees. By Exercise 8.17 #6, a $(y + 1)$-tree is either a single length-n path for some $n \in \mathbb{N}$, which we denote by $[n]$, or a ray, which we denote by $[\infty]$. So the set of objects is $\mathsf{tree}_{y+1} = \{[n] \mid n \in \mathbb{N} \cup \{\infty\}\} \cong \mathbb{N} \cup \{\infty\}$.
- A finite $(y + 1)$-tree $[n]$ for $n \in \mathbb{N}$ has $n + 1$ rooted paths: 1 each of length 0 through n, inclusive. So a morphism out of $[n]$ can be identified with an element of the set $\{0, \ldots, n\}$.
  Meanwhile, the infinite $(y + 1)$-tree $[\infty]$, a ray, has exactly 1 rooted path of every length $\ell \in \mathbb{N}$. So a morphism out of $[\infty]$ can be identified with an element of the set $\mathbb{N}$. Every $(y + 1)$-subtree of the ray is still a ray, so the codomain of each of these morphisms is still $[\infty]$.
- The identity morphism on a given B-labeled A-ary tree is its empty rooted path.
- The composite of two lists in A is obtained by concatenation.

Solution to Exercise 8.42.

1. An object in $\mathscr{T}_p$ is a stream of letters $a, b, \ldots, z$ and spaces $\sqcup$, that may go on forever or may end with a gigantic period, "$\bullet$". So an example is the infinite stream $aaaaa\cdots$. Another example is $hello\sqcup world\bullet$.
2. The object $aaaaa\cdots$ has $\mathbb{N}$ as its set of emanating morphisms. The object $hello\sqcup world\bullet$ has $\{0, \ldots, 11\}$ as its set of emanating morphisms.
3. The codomain of any morphism out of $aaaaa\cdots$ is again $aaaaa\cdots$. The codomain of any morphism $0 \le i \le 11$ out of $hello\sqcup world\bullet$ is the string one obtains by removing the first i letters of $hello\sqcup world\bullet$.

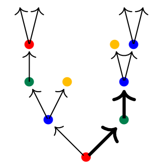
Solution to Exercise 8.43.

1.

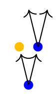


2. The identity morphism at t is the trivial path starting at the root node.

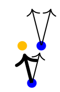
3. We indicate a morphism f using thick arrows:



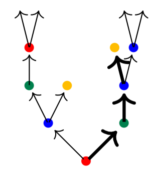
4. The codomain of f is the tree rooted at the target of the path, namely



5. Here's a morphism emanating from the codomain of f:



6. The composite morphism on the original tree is:



Solution to Exercise 8.44.

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 retrofunctor 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

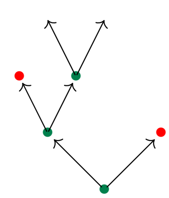
$$\varphi^\sharp_{\operatorname{cod}\varphi_i^\sharp(q_1)}(q_2) \stackrel{?}{=} \varphi_i^\sharp(q_1 + q_2)$$

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.

Solution to Exercise 8.53.

Recall that $p := \{\textcolor{green}{\bullet}\}y^2 + \{\textcolor{red}{\bullet}\}$ and $q := \{\textcolor{green}{\bullet}, \textcolor{orange}{\bullet}\}y + \{\textcolor{red}{\bullet}, \bullet\}$.

1. Have $\varphi\colon p \to q$ send forward $\textcolor{green}{\bullet} \mapsto \textcolor{green}{\bullet}$, with the unique element of $q[\textcolor{green}{\bullet}]$ sent back to the left branch of $p[\textcolor{green}{\bullet}]$, and send forward $\textcolor{red}{\bullet} \mapsto \textcolor{red}{\bullet}$.
2. Here is a sample $T \in \mathsf{tree}_p$.



3. Here is $\mathscr{T}_\varphi(T)$:

Solution to Exercise 8.54.

1. By the adjunction from Theorem 8.45, retrofunctors $\mathscr{T}_p \to \mathscr{T}_{p^{\triangleleft n}}$ are in bijection with lenses $\mathfrak{t}_p \to p^{\triangleleft n}$, where $\mathfrak{t}_p = U\mathscr{T}_p$ is the carrier.
   From Proposition 7.20 we have a lens $\delta^{(n)} \colon \mathscr{T}_p \to \mathscr{T}_p^{\triangleleft n}$, and we also have the counit of the cofree adjunction which we'll temporarily call $r \colon \mathscr{T}_p \to p$. Then the desired lens is $(\delta^{(n)} \mathbin{⨾} r^{\triangleleft n}) \colon \mathfrak{t}_p \to p^{\triangleleft n}$.
2. When $n = 1$ the induced retrofunctor is an isomorphism, but for $n \geq 2$ it is not. However, we should start by saying that the function on objects $\mathsf{tree}_p \to \mathsf{tree}_{p^{\triangleleft n}}$ is a bijection. It sends a p-tree to the $p^{\triangleleft n}$ tree that simply compresses every height-n segment into a single vertex, labeled by that height-n segment. To go back, just decompress the segments. But this is not a bijection on maps, because every rooted path on $\mathsf{tree}_{p^{\triangleleft n}}$ would correspond to a rooted path on tree_p whose length is a multiple of n. When $n = 2$ and $p = y + 1$, some rooted paths in tree_p have odd lengths.

Solution to Exercise 8.62.

A category Sy with linear carrier is a discrete category on S, so we need to show that a retrofunctor between the discrete category on S and the discrete category on T is the same thing as a function $S \to T$. But this is clear: a retrofunctor $Sy \to Ty$ is a function $S \to T$ on objects and a function backwards on morphisms, and the latter is unique because each object in Sy has a unique outgoing morphism.

Solution to Exercise 8.65.

1. Given an object $i \in \mathfrak{c}'(1) = \mathfrak{c}(1)$, the set of emanating morphisms is $\mathfrak{c}'[i] := \mathfrak{d}[fi]$, and they compose according to the structure of $\mathfrak{d}$. Since morphisms in $\mathfrak{d}$ compose associatively and unitaly, so do morphisms in $\mathfrak{c}'$.
2. Given $i \in \mathfrak{c}(1)$ and $m \in \mathfrak{c}'[i] = \mathfrak{d}[fi]$, we have $g(\mathrm{cod}(g_i^\sharp(m))) = \mathrm{cod}(g_i^\sharp(m)) = \mathrm{cod}(f_i^\sharp(m)) = \mathrm{cod}(m)$ by definition.
3. For any $i \in \mathfrak{c}'(1)$, the lens $g_i^\sharp$ preserves compositions of morphisms in $\mathfrak{c}'[i] = \mathfrak{d}[fi]$ because it agrees with $f_i^\sharp$, which preserves compositions.
4. Given $i \in \mathfrak{c}'(1)$ and $m \in \mathfrak{d}[hi]$, we have $h(\mathrm{cod}(h_i^\sharp(m))) = f(\mathrm{cod}(f_i^\sharp(m))) = m$.
5. The map $h_i^\sharp$ was chosen to be the identity for each i, so it certainly preserves compositions.

Solution to Exercise 8.68.

The category corresponding to Sy^S is the contractible groupoid K_s on S. A vertical retrofunctor $K_S \twoheadrightarrow \mathcal{C}$ includes a bijection on objects, so we can assume that $\mathrm{Ob}(\mathcal{C}) = S$. But the rest of the retrofunctor assigns to each map $s_1 \to s_2$ in $\mathcal{C}$ some choice of morphism $s_1 \to s_2$ in K_S, and there is exactly one. Thus the retrofunctor includes no additional data.

Solution to Exercise 8.69.

Let's take the two nonidentity arrows in $\mathcal{C}$ to be labeled s, t and the unique nonidentity arrow in $\mathcal{D}$ to be labeled f. Then the carrier of $\mathcal{C}$ is $\mathfrak{c} := y^{\{\mathrm{id}_1, s, t\}} + y^{\{\mathrm{id}_2\}}$ and that of $\mathcal{D}$ is $\mathfrak{d} := y^{\{\mathrm{id}_1, f\}} + y^{\{\mathrm{id}_2\}}$.

1. The boo functor $F \colon \mathcal{C} \to \mathcal{D}$ can be identified with a vertical retrofunctor $\mathfrak{d} \to \mathfrak{c}$ that sends back $\mathrm{id}_1 \mapsto \mathrm{id}_1$ and $s \mapsto f$ and $t \mapsto f$ and $\mathrm{id}_2 \mapsto \mathrm{id}_2$.
2. For $G_1 \colon \mathcal{D} \to \mathcal{C}$, which we take to send $f \mapsto s$, the corresponding vertical retrofunctor $\mathfrak{c} \to \mathfrak{d}$ sends back $\mathrm{id}_1 \mapsto \mathrm{id}_1$ and $f \mapsto s$ and $\mathrm{id}_2 \mapsto \mathrm{id}_2$.

Solution to Exercise 8.74.

1. We actually already showed that $\mathtt{0}$ has a unique comonoid structure, corresponding to the empty category (which we will also denote by $\mathtt{0}$), in Exercise 7.39, for the case of $S := \mathtt{0}$.

2. We show that 0 has the universal property of the initial object in $\mathbf{Cat}^\sharp$ as follows. For any category $\mathcal{D}$, there is a unique retrofunctor $0 \nrightarrow \mathcal{D}$: it vacuously sends each object in 0 to an object in $\mathcal{D}$, and since there are no objects in 0, it does nothing to morphisms.
   Alternatively, we know by Corollary 8.72 that $\mathbf{Cat}^\sharp$ has an initial object, and that the forgetful functor $\mathbf{Cat}^\sharp \to \mathbf{Poly}$ sends it to the initial object in $\mathbf{Poly}$, which is 0 (by Proposition 3.3 and the following discussion). So the initial object in $\mathbf{Cat}^\sharp$ must be the unique comonoid carried by 0.

Solution to Exercise 8.75.

The coproduct of $\mathfrak{c}$ with itself is again a comonoid (as a category it is the disjoint union of $\mathfrak{c}$ with itself) and it has the right carrier polynomial, since the carrier functor $U\colon \mathbf{Cat}^\sharp \to \mathbf{Poly}$ is a left adjoint.

Solution to Exercise 8.78.

1. The highbrow way to check that $\mathfrak{c} \otimes \mathfrak{d}$ forms a comonoid is to say that

$$\otimes\colon (\mathbf{Poly}, y, \triangleleft) \times (\mathbf{Poly}, y, \triangleleft) \longrightarrow (\mathbf{Poly}, y, \triangleleft)$$

   is a colax monoidal functor, so it sends comonoids to comonoids. A lowbrow way to check it is to see that the category corresponding to $\mathfrak{c} \otimes \mathfrak{d}$ is just the usual cartesian product $\mathcal{C} \times \mathcal{D}$ in $\mathbf{Cat}$. This is a category, hence a comonoid in $\mathbf{Poly}$.
2. Because $\otimes$ is symmetric, it suffices to show that for any retrofunctor $\varphi\colon \mathcal{C} \nrightarrow \mathcal{C}'$, there is an induced retrofunctor $\varphi \otimes \mathcal{D}\colon \mathcal{C} \otimes \mathcal{D} \nrightarrow \mathcal{C}' \otimes \mathcal{D}$.

Solution to Exercise 8.80.

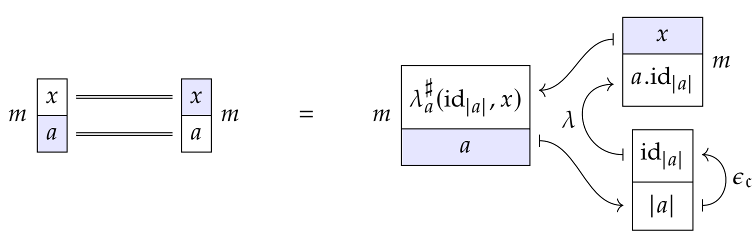
1. The carrier of $\mathcal{T}_y$ is $\mathfrak{t}_y \coloneqq y^{\mathbb{N}}$.
2. The lens $y \to \mathfrak{t}y$ is the unique lens.
3. Given a p-tree T and a q-tree U, we get a $(p \otimes q)$-tree by having its root be labeled by the pair of root labels for T and U, and for each child there—the set of which is the set of pairs consisting of a child in T and a child in U—recursively using the above formula to combine these two children.

Solution to Exercise 8.83.

In Definition 8.81, if the carrier of the left $\mathcal{C}$-comodule is always chosen to be a constant polynomial, i.e. a set, then we recover Definition 7.96. Hence constant left $\mathcal{C}$-comodules are precisely left $\mathcal{C}$-coalgebras; their notions of morphisms coincide as well, so the categories they form are isomorphic.

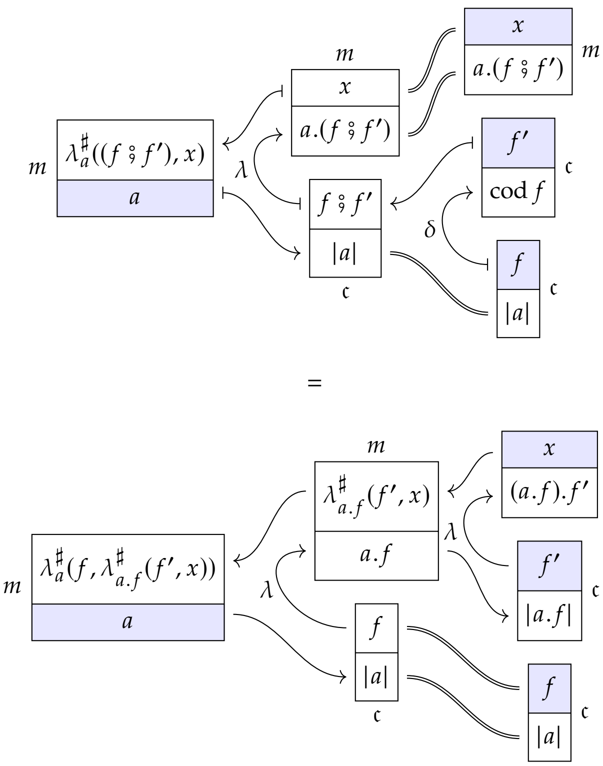
Solution to Exercise 8.86.

1. Here is the first equation in terms of polyboxes:



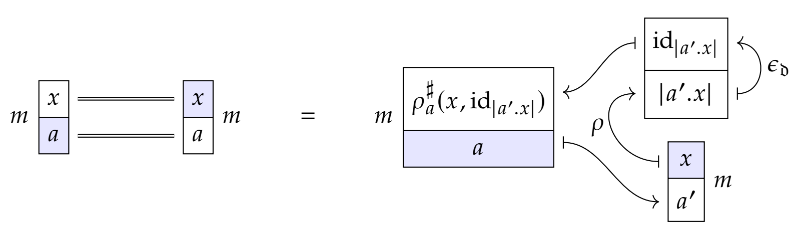
   The equation says that for any $a \in m(1)$ we have $a = a.\mathrm{id}_{|a|} \coloneqq \lambda(a, \mathrm{id}_{|a|})$ and that for any $x \in m[a]$ we have $x = x' \coloneqq \lambda_a^\sharp(\mathrm{id}_{|a|}, x)$.

Here is the second equation in terms of polyboxes:



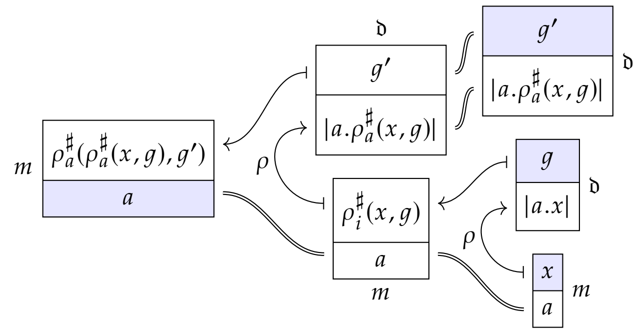
These say that $\operatorname{cod}(f) = |a.f|$, that $a.(f \mathbin{⨾} f') = (a.f).f'$, and that $\lambda_a^\sharp((f \mathbin{⨾} f'), x) = \lambda_a^\sharp(f, \lambda_{a.f}^\sharp(f', x))$.

2.

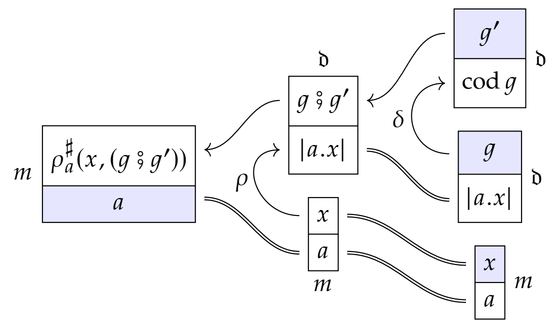


The equation says that for any $a \in m(1)$ we have $a = a' := \rho_1(i)$ and that for any $f \in m[a]$ we have $x = \rho_a^\sharp(x, \mathrm{id}_{|a'.x|})$.

Here is the second equation in terms of polyboxes:



=



These equations say that $\text{cod}(g) = |a.\rho_a^\sharp(x, g)|$ and that $\rho_a^\sharp(\rho_a^\sharp(x, g), g') = \rho_a^\sharp(x, (g \mathbin{⨟} g'))$.

Solution to Exercise 8.87.

The comonoid structure on y has $\epsilon\colon y \to y$ and $\delta\colon y \to y \triangleleft y = y$ the unique lenses.

1. We first show that for any polynomial m, there is a unique y-comodule structure on m. By (8.82), a y-comodule structure is a lens $\lambda\colon m \to y \triangleleft m = m$ such that $\lambda \mathbin{⨟} (! \triangleleft m) = \text{id}_m$, but $! \triangleleft m = \text{id}_m$, so $\lambda = \text{id}_m$ is forced. A morphism $(m, \text{id}) \to (n, \text{id}_n)$ of y-comodules is also easily seen to be just a lens $m \to n$.
2. Similar.

Solution to Exercise 8.99.

1. By Exercise 8.87, it suffices to check that the diagram (8.97) commutes vacuously when $\mathfrak{d} = y$.
2. By Exercise 8.87, it suffices to check that the diagram (8.97) commutes vacuously when $\mathfrak{c} = y$.
3. This follows from Exercise 8.87.
4. This follows from Exercise 8.87.

Solution to Exercise 8.107.

For each $v \in V$, let $A_v \coloneqq \text{src}^{-1}(v)$, the set of arrows emanating from v.

1. We need a lens $\sum_{v \in V} y^{A_v} \to V \sum_{v \in V} y^{A_v}$, and we use the obvious "diagonal" lens, which on positions sends $v \mapsto (v, v)$ and on directions is the identity.
2. We need a lens $\sum_{v \in V} y^{A_v} \to \sum_{v \in V} \prod_{a \in A_v} \sum_{v \in V} y$, which is the same as an element of

$$\prod_{v \in V} \sum_{v' \in V} \prod_{a \in A_{v'}} \sum_{v'' \in V} A_v$$

and we use $v \mapsto (v, a \mapsto \text{tgt}(a), a)$. Most of this is forced on us, and the only interesting part is the function $A_v \to V$, which we can take to be anything, the choice being exactly the choice of "target map" that defines our graph.

3. Let $T = 2V$ and take the map $2V \to V \times 2V$ to be $(i, v) \mapsto (v, i, v)$ for any $i \in 2$ and $v \in V$.

Solution to Exercise 8.110.

Recall that the $(Vy, 0)$-bicomodule T assigns to each vertex $v \in V$ some set T_v of "colors", whereas the $(Vy, 0)$-bicomodule V assigns each vertex a singleton set. A map $V \to T$ between them chooses one color for each vertex, which we're calling the "starting color".

Chapter 9

New horizons

In this brief chapter, 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 $\mathbf{Set} \to \mathbf{Set}$, e.g. ones that aren't polynomial.
2. What can you say about the internal logic for the topos $[\mathscr{T}_p, \mathbf{Set}]$ 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 **Mod**. Find examples of these comonoids, and perhaps characterize them or create a theory of them.
6. The category **Poly** has pullbacks, so one can consider the bicategory of spans in **Poly**. Is there a functor from that to **Mod** 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 $\mathbf{Set} \to \mathbf{Set}$, what happens if we replace sets with homotopy types: how much goes through? Is anything improved?
10. Are there any functors $\mathbf{Set} \to \mathbf{Set}$ 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. Describe the limits that exist in $\mathbf{Cat}^\sharp$ combinatorially.
13. 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 retrofunctor whose underlying map on carriers is monic, then it is monic. Are all monomorphisms in $\mathbf{Cat}^\sharp$ of this form?
14. Are there polynomials p such that one use something like Gödel numbers to encode logical propositions from the topos $[\mathsf{tree}_p, \mathbf{Set}]$ into a "language" that p-dynamical systems can "work with"?

Bibliography

[AAG05] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. “Containers: Constructing strictly positive types”. In: *Theoretical Computer Science* 342.1 (2005). Applied Semantics: Selected Topics, pp. 3–27 (cit. on p. 37).

[Abb+03] Michael Abbott, Thorsten Altenkirch, Neil Ghani, and Conor McBride. “Derivatives of containers”. In: *International Conference on Typed Lambda Calculi and Applications*. Springer. 2003, pp. 16–30 (cit. on p. 51).

[Abb03] Michael Gordon Abbott. “Categories of Containers”. PhD thesis. University of Leicester, Aug. 2003 (cit. on p. 37).

[Abo+16] Faris Abou-Saleh, James Cheney, Jeremy Gibbons, James McKinna, and Perdita Stevens. “Reflections on monadic lenses”. In: *A List of Successes that can Change the World*. Springer, 2016, pp. 1–31 (cit. on p. 72).

[Agu97] Marcelo Aguiar. *Internal categories and quantum groups*. Cornell University, 1997 (cit. on pp. 262, 278).

[AU16] Danel Ahman and Tarmo Uustalu. “Directed Containers as Categories”. In: *EPTCS 207, 2016, pp. 89-98* (2016). eprint: `arXiv:1604.01187` (cit. on p. 277).

[Bor94] Francis Borceux. *Handbook of categorical algebra 1*. Vol. 50. Encyclopedia of Mathematics and its Applications. Basic category theory. Cambridge University Press, Cambridge, 1994 (cit. on p. 18).

[BPS19] Erwan Beurier, Dominique Pastor, and David I Spivak. “Memoryless systems generate the class of all discrete systems”. In: *International Journal of Mathematics and Mathematical Sciences* 2019 (2019) (cit. on p. 129).

[BPV06] Aaron Bohannon, Benjamin C Pierce, and Jeffrey A Vaughan. “Relational lenses: a language for updatable views”. In: *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM. 2006, pp. 338–347 (cit. on p. 72).

[Cap22] Matteo Capucci. *Diegetic representation of feedback in open games*. 2022. URL: https://arxiv.org/abs/2206.12338 (cit. on p. 166).

[Che22] Eugenia Cheng. *The Joy of Abstraction: An Exploration of Math, Category Theory, and Life*. Cambridge University Press, 2022. DOI: 10.1017/9781108769389 (cit. on p. 18).

[Con12] John Horton Conway. *Regular algebra and finite machines*. Courier Corporation, 2012 (cit. on p. 129).

[FS19] Brendan Fong and David I. Spivak. *Seven Sketches in Compositionality: An Invitation to Applied Category Theory*. Cambridge University Press, 2019 (cit. on p. 18).

[GH18] Richard Garner and Tom Hirschowitz. "Shapely monads and analytic functors". In: *Journal of Logic and Computation* 28.1 (2018), pp. 33–83 (cit. on p. 339).

[GJ12] Jeremy Gibbons and Michael Johnson. "Relating algebraic and coalgebraic descriptions of lenses". In: *Electronic Communications of the EASST* 49 (2012) (cit. on p. 72).

[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 pp. 37, 210, 216).

[Hed+16] Jules Hedges, Evguenia Shprits, Viktor Winschel, and Philipp Zahn. *Compositionality and string diagrams for game theory*. 2016. eprint: arXiv:1604.06061 (cit. on p. 72).

[Hed17] Jules Hedges. *Coherence for lenses and open games*. 2017. eprint: arXiv:1704.02230 (cit. on p. 72).

[Hed18a] Jules Hedges. *Limits of bimorphic lenses*. 2018. eprint: arXiv:1808.05545 (cit. on pp. 56, 72).

[Hed18b] Jules Hedges. "Morphisms of open games". In: *Electronic Notes in Theoretical Computer Science* 341 (2018), pp. 151–177 (cit. on p. 72).

[Jac17] Bart Jacobs. *Introduction to Coalgebra*. Vol. 59. Cambridge University Press, 2017 (cit. on pp. 37, 208).

[JRW12] Michael Johnson, Robert Rosebrugh, and Richard J Wood. "Lenses, fibrations and universal translations". In: *Mathematical Structures in Computer Science* 22.1 (2012), pp. 25–42 (cit. on p. 72).

[Kel74] G Max Kelly. "Doctrinal adjunction". In: *Category seminar*. Springer. 1974, pp. 257–280 (cit. on p. 327).

[Koc] Joachim Kock. "Notes on Polynomial Functors". https://mat.uab.cat/~kock/cat/polynomial.pdf. Accessed 2023/12/04 (cit. on p. 37).

[Lei14] Tom Leinster. *Basic category theory*. Vol. 143. Cambridge University Press, 2014 (cit. on p. 18).

[Mac98] Saunders Mac Lane. *Categories for the working mathematician*. 2nd ed. Graduate Texts in Mathematics 5. New York: Springer-Verlag, 1998 (cit. on p. 18).

[McB01] Conor McBride. "The derivative of a regular type is its type of one-hole contexts". In: *Unpublished manuscript* (2001), pp. 74–88 (cit. on p. 51).

[MM92] Saunders MacLane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Springer, 1992 (cit. on p. 15).

[Mye22] David Jaz Myers. *Categorical Systems Theory*. 2022 (cit. on p. viii).

[nLa18] Contributors To nLab. *Created limit — nLab*. 2018. URL: `https://ncatlab.org/nlab/show/created+limit` (cit. on p. 325).

[nLa19] Contributors To nLab. *Connected limit — nLab*. 2019. URL: `https://ncatlab.org/nlab/show/connected+limit` (cit. on p. 212).

[nLa22] nLab authors. *lens (in computer science)*. `http://ncatlab.org/nlab/show/lens%20%28in%20computer%20science%29`. Revision 26. Aug. 2022 (cit. on p. 278).

[OCo11] Russell O'Connor. *Functor is to lens as applicative is to biplate: Introducing multiplate*. 2011. eprint: `arXiv:1103.2841` (cit. on p. 72).

[Par23] Robert Paré. *Retrocells*. 2023. arXiv: `2306.06436 [math.CT]` (cit. on pp. 254, 257).

[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. 325).

[Pie91] Benjamin C. Pierce. *Basic Category Theory for Computer Scientists*. MIT Press, 1991 (cit. on p. 18).

[Por19] Hans-E Porst. "Colimits of monoids". In: *Theory and Applications of Categories* 34.17 (2019), pp. 456–467 (cit. on pp. 325, 326).

[Rie17] Emily Riehl. *Category theory in context*. Courier Dover Publications, 2017 (cit. on p. 18).

[Sha21] Brandon Shapiro. *Familial Monads as Higher Category Theories*. 2021. URL: `https://arxiv.org/abs/2111.14796` (cit. on p. 339).

[Shu08] Michael Shulman. "Framed bicategories and monoidal fibrations". In: *Theory and Applications of Categories* 20 (2008), Paper No. 18, 650–738 (cit. on pp. 163, 165, 166).

[Spi12] David I. Spivak. "Functorial data migration". In: *Information and Computation* 217 (2012), pp. 31–51 (cit. on p. 274).

[Spi19] David I. Spivak. *Generalized Lens Categories via functors* $\mathcal{C}^{op} \to \mathsf{Cat}$. 2019. eprint: arXiv:1908.02202 (cit. on p. 72).

[Spi22] David I. Spivak. *Polynomial functors and Shannon entropy*. 2022. arXiv: 2201.12878 [math.CT] (cit. on p. 166).

[Spi23] David I. Spivak. *Functorial aggregation*. 2023. arXiv: 2111.10968 [math.CT] (cit. on p. vii).

[ST17] David I Spivak and Joshua Tan. "Nesting of dynamical systems and mode-dependent networks". In: *Journal of Complex Networks* 5.3 (2017), pp. 389–408 (cit. on p. 129).

[SW15] David I. Spivak and Ryan Wisnesky. "Relational Foundations for Functorial Data Migration". In: *Proceedings of the 15th Symposium on Database Programming Languages*. DBPL. Pittsburgh, PA: ACM, 2015, pp. 21–28 (cit. on p. 336).

[Web07] Mark Weber. "Familial 2-Functors and Parametric Right Adjoints". In: *Theory and Applications of Categories* 18 (2007), Paper No. 22, 665–732 (cit. on p. 339).

Index

Number of todo notes in text, at last pass: 0