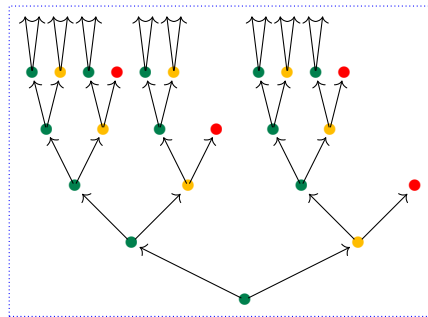


# Polynomial Functors: A Mathematical Theory of Interaction



Nelson Niu

David I. Spivak

Last updated: June 12, 2023

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*

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

—Frank Wilczek  
*Fundamentals*

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.

## 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. Do not be discouraged if you encounter words or ideas that we do not thoroughly explain—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.

## 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.

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

The natural numbers include 0, so  $\mathbb{N} := \{0, 1, 2, \dots\}$ .

The names of categories will be capitalized. We will not focus so much on 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. **Poly**, **C**). We use **Set** to denote the category of (small) sets and functions and **Cat** to denote the category of (small) categories and functors. We use exponential notation  $\mathcal{D}^{\mathcal{C}}$  to denote the category of functors  $\mathcal{C} \rightarrow \mathcal{D}$  and natural transformations.

We write either  $c \in \text{Ob } \mathcal{C}$  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  $f: c \rightarrow d$  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  $\text{dom } f$  and the codomain of  $f$  by  $\text{cod } f$ . 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 generally indicate anything special 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 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/>.

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 as we were writing this, we were discovering 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, 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.



---

# Contents

---

<b>I</b>	<b>The category of polynomial functors</b>	<b>1</b>
<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Perspectives on polynomials . . . . .	3
1.2	Decisions and interaction in series . . . . .	6
1.3	Dynamical systems . . . . .	7
1.4	A note on implementation . . . . .	11
1.5	Mathematical theory . . . . .	11
1.6	Exercise solutions . . . . .	14
<b>2</b>	<b>Polynomial functors and natural transformations</b>	<b>15</b>
2.1	Representable functors and the Yoneda lemma . . . . .	15
2.2	Polynomials: sums of representables . . . . .	18
2.2.1	Sums and products of sets . . . . .	19
2.2.2	Expanding products of sums . . . . .	22
2.2.3	Dependent sums and products of functors <b>Set</b> $\rightarrow$ <b>Set</b> . . . . .	24
2.2.4	What is a polynomial functor? . . . . .	27
2.3	Morphisms between polynomial functors . . . . .	29
2.3.1	Coproducts of polynomials . . . . .	29
2.3.2	Polynomial morphisms, concretely . . . . .	30
2.3.3	Arena morphisms as dependent lenses . . . . .	35
2.3.4	Translating between natural transformations and lenses . . . . .	37
2.3.5	Identity and composition of lenses . . . . .	39
2.4	Symmetric monoidal products of polynomial functors . . . . .	41
2.4.1	The categorical product . . . . .	41
2.4.2	The parallel product . . . . .	43
2.5	Summary and further reading . . . . .	47
<b>3</b>	<b>Dynamical systems as dependent lenses</b>	<b>49</b>
3.1	Moore machines . . . . .	49
3.1.1	Moore machines as lenses . . . . .	50

3.1.2	More Moore machines . . . . .	51
3.2	Dependent dynamical systems . . . . .	54
3.2.1	Thinking about dependency . . . . .	55
3.2.2	Examples of dependent dynamical systems . . . . .	56
3.3	Constructing new dynamical systems from old . . . . .	60
3.3.1	Categorical products: multiple interfaces operating on the same states . . . . .	61
3.3.2	Parallel products: juxtaposing dynamical systems . . . . .	62
3.3.3	Composing lenses: wrapper interfaces . . . . .	65
3.3.4	Situations as enclosures . . . . .	66
3.4	General interaction . . . . .	69
3.4.1	Wrapping juxtaposed dynamical systems together . . . . .	69
3.4.2	Enclosing juxtaposed dynamical systems together . . . . .	73
3.4.3	Wiring diagrams as interaction patterns . . . . .	77
3.4.4	More examples of general interaction . . . . .	81
3.5	Closure of $\otimes$ . . . . .	85
3.6	Summary and further reading . . . . .	89
<b>4</b>	<b>More categorical properties of polynomials</b>	<b>91</b>
4.1	Special polynomials and adjunctions . . . . .	91
4.2	Epi-mono factorization of lenses . . . . .	94
4.3	Cartesian closure . . . . .	98
4.4	Limits and colimits of polynomials . . . . .	99
4.5	Vertical-cartesian factorization of lenses . . . . .	104
4.6	Monoidal $*$ -bifibration over <b>Set</b> . . . . .	108
4.7	Summary and further reading . . . . .	112
<b>II</b>	<b>A different category of categories</b>	<b>113</b>
<b>5</b>	<b>The composition product</b>	<b>115</b>
5.1	Defining the composition product . . . . .	115
5.1.1	Composite functors . . . . .	115
5.1.2	Composite arenas . . . . .	119
5.1.3	Composite corolla forests . . . . .	121
5.1.4	Dynamical systems and the composition product . . . . .	126
5.2	Lenses to composites . . . . .	130
5.2.1	Lenses as polyboxes . . . . .	130
5.2.2	Situations as polyboxes . . . . .	133
5.2.3	Lenses to composites as polyboxes . . . . .	135
5.2.4	The composition product of lenses as polyboxes . . . . .	139
5.3	Categorical properties of the composition product . . . . .	145

5.3.1	Interaction with products and coproducts . . . . .	145
5.3.2	Interaction with limits on the left . . . . .	147
5.3.3	Interaction with limits on the right . . . . .	150
5.3.4	Interaction with parallel products . . . . .	154
5.3.5	Interaction with vertical and cartesian lenses . . . . .	156
5.4	Summary and further reading . . . . .	156
5.5	Exercise solutions . . . . .	157
<b>6</b>	<b>Polynomial comonoids and cofunctors</b>	<b>169</b>
6.1	State systems, categorically . . . . .	169
6.1.1	The do-nothing enclosure . . . . .	170
6.1.2	The transition lens . . . . .	171
6.1.3	The do-nothing enclosure coheres with the transition lens . . . .	172
6.1.4	The transition lens is coassociative . . . . .	175
6.1.5	Running dynamical systems . . . . .	177
6.1.6	State systems as comonoids . . . . .	179
6.2	Polynomial comonoids are categories . . . . .	183
6.2.1	Translating between polynomial comonoids and categories . . . .	184
6.2.2	Examples of categories as comonoids . . . . .	190
6.3	Morphisms of polynomial comonoids are cofunctors . . . . .	196
6.3.1	Introducing comonoid morphisms and cofunctors . . . . .	196
6.3.2	Examples of cofunctors . . . . .	200
6.3.3	Equivalent characterizations of cofunctors from state categories .	211
6.4	Summary and further reading . . . . .	218
6.5	Exercise solutions . . . . .	218
<b>7</b>	<b>Categorical properties of polynomial comonoids</b>	<b>231</b>
7.1	Cofree comonoids . . . . .	231
7.1.1	The carrier of the cofree comonoid . . . . .	232
7.1.2	Cofree comonoids as categories . . . . .	243
7.1.3	Exhibiting the forgetful-cofree adjunction . . . . .	254
7.1.4	The many (inter)faces of the cofree comonoid . . . . .	256
7.1.5	Morphisms between cofree comonoids . . . . .	259
7.1.6	Some categorical properties of cofree comonoids . . . . .	259
7.2	More categorical properties of $\mathbf{Cat}^\#$ . . . . .	261
7.2.1	Other special comonoids and adjunctions . . . . .	261
7.2.2	Vertical-cartesian factorization of cofunctors . . . . .	262
7.2.3	Limits and colimits of comonoids . . . . .	264
7.2.4	Parallel product comonoids . . . . .	266
7.3	Comodules over polynomial comonoids . . . . .	267
7.3.1	Left and right comodules . . . . .	267
7.3.2	Bicomodules . . . . .	271

7.3.3	More equivalences . . . . .	273
7.3.4	Bicomodules are parametric right adjoints . . . . .	276
7.3.5	Bicomodules in dynamics . . . . .	276
7.4	Summary and further reading . . . . .	278
7.5	Exercise solutions . . . . .	279
<b>8</b>	<b>New horizons</b>	<b>289</b>
	<b>Bibliography</b>	<b>291</b>

## **Part I**

# **The category of polynomial functors**



## Chapter 1

---

# Introduction

---

It is a treasury box!  
 Full of unexpected connections!  
 It is fascinating!  
 I will think about it.

—André Joyal, Summer 2020,  
 personal communication.

## 1.1 Perspectives on polynomials

In this book we will investigate a remarkable category called **Poly** and its intimate relationships with interactive and dynamic processes. But our story begins with something quite humble—high school algebra:

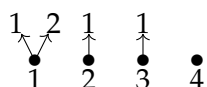
$$y^2 + 2y + 1 \quad \text{polynomial} \tag{1.1}$$

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

$$\begin{array}{c} \swarrow \uparrow \\ \bullet \end{array} \quad \begin{array}{c} \uparrow \\ \bullet \end{array} \quad \begin{array}{c} \uparrow \\ \bullet \end{array} \quad \bullet \quad \text{forest} \tag{1.2}$$

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

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



This suggests yet another depiction of the polynomial (1.1): as an *indexed family of sets*  $(p[i])_{i \in I}$ . Here  $I = 4 = \{1, 2, 3, 4\}$ <sup>1</sup>, the set of roots, and

$$p[1] = 2 = \{1, 2\}, \quad p[2] = 1 = \{1\}, \quad p[3] = 1 = \{1\}, \quad p[4] = 0 = \emptyset, \quad \text{arena} \quad (1.3)$$

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

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

#### Polynomial Terminology

<b>algebra</b>	$p := \sum_{i \in p(1)} y^{p[i]}$	$i \in p(1)$	$a \in p[i]$	
<b>dependent sets</b>	arena	position	direction	
<b>tree pictures</b>	corolla forest	root •	leaf ↑	
<b>decisions</b>	scenario	menu	option	
<b>dynamics</b>	interface	output	input	(1.4)

We will freely switch between these equivalent terms depending on context. For instance, though a polynomial will turn out to be a *functor*, while an arena is a *dependent set*, they are so closely related that we often do not make a distinction between a polynomial  $p$  and its arena  $(p[i])_{i \in I}$ ; they are two different syntaxes for the same object. In particular, we will directly refer to the positions and directions of a polynomial when we mean the positions and directions of its associated arena.

**Exercise 1.5.** Consider the polynomial  $p := 2y^3 + 2y + 1$  and the associated corolla forest and arena.

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

<sup>1</sup>In standard font, 4 represents the usual natural number. In sans serif font, 4 represents the set  $4 = \{1, 2, 3, 4\}$  with 4 elements.



*Exercise 1.6.* Consider the polynomial  $q := y^8 + 4y$ .

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

◇

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



*Exercise 1.7.* If you were a suitor choosing the corolla forest you love, aesthetically speaking, which would strike your interest? Answer by circling the associated polynomial:

1.  $y^2 + y + 1$
2.  $y^2 + 3y^2 + 3y + 1$
3.  $y^2$
4.  $y + 1$
5.  $(\mathbb{N}y)^{\mathbb{N}}$
6.  $Sy^S$
7.  $y^{100} + y^2 + 3y$
8.  $y + 2y^4 + 3y^9 + 4y^{16} + \dots$
9. Your polynomial's name  $p$  here.

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

◇

Before we can really get into this story, let's summarize where we're going: polynomials are going to have really surprising applications to modeling general interaction, and **Poly** as a category has an abundance of structure that we can draw on. We speak superlatively of **Poly**:

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

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

## 1.2 Decisions and interaction in series

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

$$\begin{array}{c} \nwarrow \nearrow \quad \uparrow \quad \uparrow \quad \cdot \\ \text{green} \quad \text{blue} \quad \text{yellow} \quad \text{red} \end{array} \quad (1.8)$$

As in our arena language, we will call the corollas in the forest for  $p$  the  $p$ -corollas, whose roots are  $p$ -roots and whose leaves at each root  $i$  are the  $p[i]$ -leaves. Since each corolla can be identified with its root, we will often use the same name  $i$  for both the  $p$ -corolla and its  $p$ -root.

Concretely, we might think of each position as representing a *menu*. Associated to every menu is a set of *options* (directions). The four menus we exhibit in (1.8) are particularly interesting: they respectively have two options, one option, one option, and no options. Having two options is familiar from life—it's the classic yes/no decision—as well as from Claude Shannon's information theory. Having one option is also familiar both theoretically and in life: "sorry, ya just gotta go through it." Having no options is when you actually don't get through it: an impossible decision, a sort of "dead end." We refer to the menus and options represented by the arena as a whole as the *scenario*.

This decision-making perspective will prove insightful when we start to discuss the *morphisms* between two polynomials. A morphism will model how two different agents, each in their own scenario, may interact. If we think of our lives as series of decisions we make, we interact with others by letting our own decisions depend on the decisions of others. In some sense, any interaction between two agents is a way of *delegating* decisions from one scenario to another. We'll cover this in more detail in Section 2.3.

For now, let's hone in on that "in series" part a bit. Life is not a single decision—it is a series of decisions, each dependent on the last. A game theorist might model such a series with a *decision tree*, like the ones below (the first two are infinite, but that's hard to draw, so we've included just the first five levels):



Each tree models a series of decisions that may be presented to us. We start at the bottom—the root—and climb up the tree, making a choice of which arrow to follow at each step. Sometimes we are presented with two options; sometimes just one; and sometimes no option at all. Eventually, our choices trace out a path up the tree.

But squint at these trees, and you'll see that they are made up of something quite familiar—every tree is built out of several smaller corollas, of exactly the form we drew in (1.8)! It makes sense now why we interpret our polynomials both as roots and leaves of corollas and as menus and options of decisions—when we arrange these corollas and decisions in series, we get a decision tree. We call each tree that we can build out of the corollas of  $y^2 + 2y + 1$ , like the three trees drawn above, a  $(y^2 + 2y + 1)$ -tree. Similarly, we could define and construct a  $p$ -tree for any polynomial  $p$ .

What's amazing is that this is not just an arbitrarily imposed reading of our polynomials—it is directly from the categorical structure of **Poly** that such trees arise. But we will have to wait before we have all the machinery we need to discuss what's happening here—we'll revisit these trees in Section 7.1.

*Exercise 1.9.*    1. Draw the first three levels of a  $(y^2 + y^0)$ -tree.  
                     2. Draw the first four levels of a  $y$ -tree.  
                     3. Draw a the first three levels of a  $\mathbb{N}y^2$ -tree by labeling every node with a natural number. ◇

## 1.3 Dynamical systems

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

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

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

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

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

from and send output to other systems in a particular interaction pattern:



Of course, there is a lot going on in the world around us that we haven't drawn. We each have some input ports: our eyes, our ears, etc., and some output ports: our speech, our gestures, etc. We can connect with other systems: our family, our colleagues, etc. And we can think of all of these systems as subsystems of one larger system interacting with the world.



(1.10)

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

*the way systems connect can change over time.*

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

*Example 1.11.* Here are some familiar circumstances where we see interaction patterns changing over time.

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



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

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



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

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



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

All of the examples discussed here will be presented in some detail once we have the requisite mathematical theory (Examples 3.73, 3.75, and 3.76).

*Exercise 1.12.* Think of another example where systems are sending each other information, but where who the information is being sent to or received from can change based on the states of the systems involved. You might have more than two, say  $\mathbb{R}$ -many, different interaction patterns in your setting.  $\diamond$

But there's more that's intuitively wrong or limiting about the picture in (1.10). Ever notice how you can change how you interface with the world? Sometimes I close my eyes, which makes that particular way of sending me information inaccessible: that port vanishes, and you need to use your words. Sometimes I'm in a tunnel and my car can't receive a radio signal. Sometimes I extend my hand to give or receive an object from another person, but sometimes I don't. So

*a system's interface itself can change over time.*

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

And there's even more that's wrong with the above description. Namely, when I use my muscles or mouth to express things, my very position changes: my tongue

moves, my body moves. The display of an analog clock is literally the positions of its hands. So

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

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

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

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

*Example 1.13.* Imagine a million little eyeballs, each of which has a tiny brain inside it, all together in a pond of eyeballs. All that an individual eyeball  $e$  can do is open and close. When  $e$  is open, it can make some distinction about all the rest of the eyeballs in view: maybe it can count how many are open, or maybe it can see whether just one certain eyeball  $e'$  is open or closed. But when  $e$  is closed, it can’t see anything; whatever else is happening, it’s all the same to  $e$ . All it can do in that state is process previous information.

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

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

Talk about real-world applications!

*Exercise 1.14.* Give another example of a system where the range of possible inputs the system can receive is dependent on what output the system is currently providing.  $\diamond$

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

*Remark 1.15.* We ended Example 1.13 by joking about “real-world applications,” because a pond of eyeballs is about the most bizarre thing one can imagine. But recall Nobel physicist Frank Wilczek’s quote from the preface:

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

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

## 1.4 A note on implementation

In theory, everything we will discuss could be rigorously implemented and verified on a computer—as long as you have access to a functional programming language that supports dependent types, such as Agda or Idris. One implementation of **Poly** that uses Cubical Agda can be found at [Lia22]. But even in languages that are not strictly functional, the theory of **Poly** may still guide programming paradigms for the design and execution of interactive systems. In this book, we’ll focus on the math, but we encourage you to explore the field of computational implementations if that is where your interests lie.

If you do decide to embark on a computational journey, a word of warning: what we have been calling polynomials—things like  $y^2 + 2y + 1$ —are often called *containers* in the computer science (and particularly functional programming) literature. A container consists of a type  $S$ , usually called the type of *shapes*, and a type  $P(s)$  for each term  $s : S$ , called the type of *positions* in shape  $s$ . It’s mildly unfortunate that the names clash with our own: for us a container-shape is a position and a container-position is a direction.

## 1.5 Mathematical theory

The applications of **Poly** are quite diverse and interesting, encapsulating decision-based interactions and dynamical systems. However it is how the mathematics of **Poly** supports these applications that is so fantastic. For reference, we list some of the major results about **Poly** that we will cover in this book; note that this list is by no means comprehensive.

**Proposition 1.16.** **Poly** has all products and coproducts and is completely distributive. **Poly** also has exponential objects, making it a bicartesian closed category. It therefore supports the simply typed lambda calculus.

*Proof.* We will prove that **Poly** has coproducts in Proposition 2.51, that it has products in Proposition 2.80, that it is completely distributive in Corollary 2.82, and that it has exponential objects in Theorem 4.30.  $\square$

**Proposition 1.17.** Beyond the cocartesian and cartesian monoidal structures  $(0, +)$  and  $(1, \times)$ , the category **Poly** has two additional monoidal structures, denoted  $(y, \otimes)$  and  $(y, \circ)$ , which are together duoidal.<sup>a</sup> Moreover  $\otimes$  is a closed monoidal structure that distributes over coproducts, while  $\circ$  is a left coclosed monoidal structure that preserves connected limits.

<sup>a</sup>We will follow the convention of writing the tensor unit before the tensor product when specifying a monoidal structure.

*Proof.* We will define  $\otimes$  in Definition 2.89 and prove that  $(y, \otimes)$  is a monoidal structure on **Poly** in Proposition 2.97, and we will define  $\circ$  in Definition 5.1 and prove that  $(y, \circ)$  is a monoidal structure on **Poly** in Corollary 5.5. Then we will show that  $\circ$  is duoidal over  $\otimes$  in Proposition 5.93.

In Proposition 2.102, we will show that  $\otimes$  distributes over coproducts. Then in Proposition 3.88, we will prove that  $\otimes$  is closed. In Proposition 5.63, we will show that  $\circ$  is left coclosed, and in Theorem 5.86, we will show that  $\circ$  preserves connected limits.  $\square$

**Proposition 1.18.** **Poly** has an adjoint quadruple with **Set** and an adjoint pair with **Set**<sup>op</sup>:

$$\begin{array}{ccc} \text{Set} & \begin{array}{c} \xleftarrow{p(0)} \\ \xrightarrow{A} \\ \xleftarrow{p(1)} \\ \xrightarrow{Ay} \end{array} & \text{Poly} \end{array} \qquad \begin{array}{ccc} \text{Set}^{\text{op}} & \begin{array}{c} \xrightarrow{y^A} \\ \xleftarrow{\Gamma(p)} \end{array} & \text{Poly} \end{array} .^a$$

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

<sup>a</sup>We use the notation  $\mathcal{C} \xrightleftharpoons[L]{L} \mathcal{D}$  to denote an adjunction  $L \dashv R$ . The double arrow, always pointing in the same direction as the left adjoint, indicates both the unit  $\mathcal{C} \Rightarrow R \circ L$  and the counit  $L \circ R \Rightarrow \mathcal{D}$  of the adjunction.

*Proof.* We will prove that **Poly** has an adjoint quadruple with **Set** in Theorem 4.4, and that it has an adjoint pair with **Set**<sup>op</sup> in Proposition 4.12.  $\square$

**Proposition 1.19.** **Poly** has all limits and colimits.

*Proof.* See Theorem 4.32 for a proof that **Poly** has all limits, and Theorem 4.42 for a proof that **Poly** has all colimits.  $\square$



There's a lot we're leaving out of this summary, just so we can hit the highlights. But here's where things get really interesting.

**Proposition 1.20** (Ahman-Uustalu). There is a one-to-one isomorphism-preserving correspondence between comonoids in  $(\mathbf{Poly}, y, \circ)$  and small categories.

*Proof.* See Theorem 6.28. □

**Proposition 1.21.** The forgetful functor  $U: \mathbf{Comon}(\mathbf{Poly}) \rightarrow \mathbf{Poly}$  has a right adjoint

$$\mathbf{Poly} \begin{array}{c} \xrightarrow{\mathcal{T}_-} \\ \xleftarrow{U} \end{array} \mathbf{Comon}(\mathbf{Poly})$$

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

*Proof.* See Theorem 7.45; lax monoidality is proven in Proposition 7.79. □

**Proposition 1.22.** The category  $\mathbf{Comon}(\mathbf{Poly})$  of comonoids in  $(\mathbf{Poly}, y, \circ)$  has all limits and colimits. In particular, coproducts in  $\mathbf{Comon}(\mathbf{Poly})$  agree with those in  $\mathbf{Cat}$ .

*Proof.* See Corollary 7.76 for a proof that  $\mathbf{Comon}(\mathbf{Poly})$  has all small limits and Corollary 7.72 for a proof that  $\mathbf{Comon}(\mathbf{Poly})$  has all small colimits. We explain why coproducts in  $\mathbf{Comon}(\mathbf{Poly})$  agree with those in  $\mathbf{Cat}$  in Example 7.73. □

**Proposition 1.23.** The category  $\mathbf{Comon}(\mathbf{Poly})$  has a symmetric monoidal structure  $(y, \otimes)$  making the forgetful functor  $U: (\mathbf{Comon}(\mathbf{Poly}), y, \otimes) \rightarrow (\mathbf{Poly}, y, \otimes)$  strong monoidal. The monoidal product of comonoids in  $\mathbf{Comon}(\mathbf{Poly})$  with respect to  $\otimes$  coincides with the product of categories in  $\mathbf{Cat}$ .

*Proof.* See Proposition 7.77. □

For the following results, let  $\mathcal{C}$  be the comonoid in  $(\mathbf{Poly}, y, \circ)$  that corresponds to a category  $\mathcal{C}$  under Proposition 1.20, and similarly for  $\mathcal{D}$  and  $\mathcal{D}$ .

**Proposition 1.24.** The category of left  $\mathcal{C}$ -comodules is equivalent to the category of functors  $\mathcal{C} \rightarrow \mathbf{Poly}$ . Meanwhile, a right  $\mathcal{D}$ -comodule with polynomial carrier  $m$  can be identified (up to isomorphism) with a functor  $\mathcal{D} \rightarrow \mathbf{Set}^{m(1)}$ .

*Proof.* See Proposition 7.88 for the result about left  $\mathcal{C}$ -comodules and Proposition 7.89 for the result about right  $\mathcal{D}$ -comodules. □

**Proposition 1.25** (Garner). There is an equivalence

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

between the category of bicomodules over comonoids in **Poly** and parametric right adjoints between copresheaf categories.

*Proof.* See Proposition 7.104. □

If you skipped over any of that—or all of that—it’ll be no problem whatsoever! We will cover each of the above results in detail over the course of this book. As you read, we encourage you to periodically check back to see how many more of these results you understand.

There are many avenues for study, but we need to push forward. We’ll begin in the next chapter.

## 1.6 Exercise solutions

## Chapter 2

---

# Polynomial functors and natural transformations

---

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

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

### 2.1 Representable functors and the Yoneda lemma

Representable functors—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 will focus on representable functors **Set**  $\rightarrow$  **Set**.

**Definition 2.1** (Representable functor). For a set  $S$ , we denote by  $y^S: \mathbf{Set} \rightarrow \mathbf{Set}$  the functor that sends each set  $X$  to the set  $X^S = \mathbf{Set}(S, X)$  and each function  $h: X \rightarrow Y$  to the function  $h^S: X^S \rightarrow Y^S$  that sends  $g: S \rightarrow X$  to  $g \circ h: S \rightarrow Y$ .<sup>a</sup>

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*.

<sup>a</sup>Throughout this text, given morphisms  $f: A \rightarrow B$  and  $g: B \rightarrow C$  in a category, we will denote their composite morphism  $A \rightarrow C$  interchangeably as  $f \circ g$  or  $g \circ f$ , whichever is more convenient.

The symbol  $y$  stands for Yoneda, for reasons we will explain in Lemma 2.11 and Exercise 2.13 #5. For now, here are some pictures corresponding to various representables or pure powers:

$$y^5 \quad y^{10} \quad y^{20} \quad y^{40} \quad y^{[0,1]} \quad (2.2)$$

*Example 2.3.* The functor that sends each set  $X$  to  $X \times X$  and each function  $h: X \rightarrow Y$  to  $(h \times h): (X \times X) \rightarrow (Y \times Y)$  is representable. After all,  $X \times X \cong X^2$ , so this functor is the pure power  $y^2$ .

*Exercise 2.4.* For each of the following functors  $\mathbf{Set} \rightarrow \mathbf{Set}$ , 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  $X \mapsto 2$ , which sends every function to the identity on 2.
3. The constant functor  $X \mapsto 1$ , which sends every function to the identity on 1.
4. The constant functor  $X \mapsto 0$ , which sends every function to the identity on 0.
5. A functor  $X \mapsto X^{\mathbb{N}}$ . If it could be representable, where should it send each function?
6. A functor  $X \mapsto 2^X$ . If it could be representable, where should it send each function?  $\diamond$

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 2.5.** For any function  $f: R \rightarrow S$ , there is an induced natural transformation  $y^f: y^S \rightarrow y^R$ ; on any set  $X$  its  $X$ -component  $X^f: X^S \rightarrow X^R$  is given by sending  $g: S \rightarrow X$  to  $f \circ g: R \rightarrow X$ .

*Proof.* See Exercise 2.6.  $\square$

*Exercise 2.6.* To prove Proposition 2.5, show that for any function  $f: R \rightarrow S$ , the given construction  $y^f: y^S \rightarrow y^R$  really is a natural transformation. That is, for any function

$h: X \rightarrow Y$ , show that the following naturality square commutes:

$$\begin{array}{ccc} X^S & \xrightarrow{h^S} & Y^S \\ X^f \downarrow & ? & \downarrow Y^f \\ X^R & \xrightarrow{h^R} & Y^R \end{array} \quad (2.7)$$

◇

*Exercise 2.8.* Let  $X$  be an arbitrary set. For each of the following sets  $R, S$  and functions  $f: R \rightarrow S$ , describe the  $X$ -component  $X^f: X^S \rightarrow X^R$  of the natural transformation  $y^f: y^S \rightarrow y^R$ .

1.  $R = 5, S = 5, f = \text{id}_5$ . (You should describe the function  $X^{\text{id}_5}: X^5 \rightarrow X^5$ .)
2.  $R = 2, S = 1, f$  is the unique function.
3.  $R = 1, S = 2, f(1) = 1$ .
4.  $R = 1, S = 2, f(1) = 2$ .
5.  $R = 0, S = 5, f$  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  $\mathbf{Set} \rightarrow \mathbf{Set}$  and whose morphisms are the natural transformations between them.

*Exercise 2.9.* Show that the construction  $f \mapsto y^f$  from Proposition 2.5 defines a functor

$$y^-: \mathbf{Set}^{\text{op}} \rightarrow \mathbf{Set}^{\mathbf{Set}} \quad (2.10)$$

by verifying functoriality, as follows.

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

◇

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

**Lemma 2.11** (Yoneda lemma). Given a functor  $F: \mathbf{Set} \rightarrow \mathbf{Set}$  and a set  $S$ , there is an isomorphism

$$F(S) \cong \mathbf{Set}^{\mathbf{Set}}(y^S, F) \quad (2.12)$$

where the right hand side is the set of natural transformations  $y^S \rightarrow F$ . Moreover, (2.12) is natural in both  $S$  and  $F$ .

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

Conversely, given an element  $a \in F(S)$ , there is a natural transformation we denote by  $m^a: y^S \rightarrow F$  whose  $X$ -component is the function  $X^S \rightarrow F(X)$  that sends  $g: S \rightarrow X$  to  $F(g)(a)$ . In Exercise 2.13 we ask you to show that this is indeed natural in  $X$ ; that these two constructions,  $m \mapsto m_S(\text{id}_S)$  and  $a \mapsto m^a$ , are mutually inverse; and that the resulting isomorphism is natural.  $\square$

*Exercise 2.13.* In this exercise, we fill in the details of the preceding proof.

1. Show that for any  $a \in F(S)$ , the maps  $X^S \rightarrow F(X)$  defined in the proof of Lemma 2.11 are natural in  $X$ .
2. Show that the two mappings given in the proof of Lemma 2.11 are mutually inverse, thus defining the isomorphism (2.12).
3. Show that (2.12) as defined is natural in  $F$ .
4. Show that (2.12) as defined is natural in  $S$ .
5. As a corollary of Lemma 2.11, show that the functor  $y^-: \mathbf{Set}^{\text{op}} \rightarrow \mathbf{Set}^{\mathbf{Set}}$  from (2.10) 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*.  $\diamond$

## 2.2 Polynomials: sums of representables

In algebra, a polynomial is just a sum of pure powers. So we will define a *polynomial functor*  $\mathbf{Set} \rightarrow \mathbf{Set}$  to be a sum of pure power functors—that is, the representable functors  $y^A$  for each set  $A$  we just introduced.<sup>1</sup>

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 following bizarre polynomial

$$\mathbb{R}y^{\mathbb{Z}} + 3y^3 + y^A + \sum_{i \in I} y^{R_i + Q_i^2}, \quad (2.14)$$

$\mathbb{R}$  denotes the set of real numbers,  $\mathbb{Z}$  denotes the set of integers, 2 and 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 (2.14). In particular, just as  $3y^3 = y^3 + y^3 + y^3$  in algebra, the summand  $3y^3$  of (2.14) denotes the sum of representables  $y^3 + y^3 + y^3$ , 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

<sup>1</sup>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.

sense of these products, we will define functor multiplication as well. The construction of sums and products of functors  $\mathbf{Set} \rightarrow \mathbf{Set}$  will rely on the construction of sets and products of sets themselves.

### 2.2.1 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

A single element from one set in the collection would be denoted  $(i, x)$  where  $x \in X_i$ . Choosing an element from every set in the collection would give us a function  $i \mapsto x_i$ , where each  $x_i \in X_i$ . This is a sort of function we haven't seen before, at least in this form—its codomain *depends* on the element we are applying it to. Think of a vector field  $v$ : to each point  $p$  it assigns a tangent vector  $v_p$  in the tangent space *at*  $p$ . We can write the signature of such a function as

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

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

**Definition 2.15** (Dependent sum and product of sets). Let  $I$  be a set and  $X: I \rightarrow \mathbf{Set}$  be an  $I$ -indexed collection of sets. The (*dependent*) *sum*  $\sum_{i \in I} X_i$  and (*dependent*) *product*  $\prod_{i \in I} X_i$  of this collection are the sets

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

*Example 2.16.* If  $I = 2 = \{1, 2\}$  then an  $I$ -indexed family  $X: I \rightarrow \mathbf{Set}$  is just two sets, say  $X_1 = \{a, b, c\}$  and  $X_2 = \{c, d\}$ . Their sum is the disjoint union

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

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

Meanwhile, their product is the usual cartesian product

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

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

*Exercise 2.17.* Let  $I$  be a set and let  $X_i := 1$  be a one-element set for each  $i \in I$ .

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

As a special case, suppose  $I := \emptyset$  and  $X: \emptyset \rightarrow \mathbf{Set}$  is the unique empty indexed family of sets.

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

We’ll assume you’re already familiar with the following fact, which justifies why we call the constructions in Definition 2.15 sums and products.

**Proposition 2.18.** Let  $I$  be a set and  $X: I \rightarrow \mathbf{Set}$  be an  $I$ -indexed family of sets. Then the sum  $\sum_{i \in I} X_i$  is the categorical coproduct of these sets in  $\mathbf{Set}$  (i.e. the colimit of the functor  $X: I \rightarrow \mathbf{Set}$ ), and the product  $\prod_{i \in I} X_i$  is the categorical product of these sets in  $\mathbf{Set}$  (i.e. the limit of the functor  $X: I \rightarrow \mathbf{Set}$ ).

*Exercise 2.19.* Let  $X: I \rightarrow \mathbf{Set}$  be a set depending on an  $i \in I$ . There is a projection function  $\pi_1: \sum_{i \in I} X_i \rightarrow I$  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  $r: A \rightarrow B$  is a function  $s: B \rightarrow A$  such that  $s \circ r = \text{id}_B$ . Show that the dependent product is isomorphic to the set of sections of  $\pi_1$ :

$$\prod_{i \in I} X_i \cong \left\{ s: I \rightarrow \sum_{i \in I} X_i \mid s \circ \pi_1 = \text{id}_I \right\}.$$

◇

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

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  $\pi_1$  from Exercise 2.19 sends each element of  $\sum_{i \in I} X_i$  to the element of  $i \in I$  chosen in step 1, while the projection  $\pi_2$  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

$$A := \sum_{i \in I} \prod_{j \in J(i)} \sum_{k \in K(i,j)} X(i, j, k). \quad (2.20)$$

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

To choose an element of  $A$ :

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

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

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

*Example 2.21.* Let  $I = \{1, 2\}$ ; let  $J(1) = \{j\}$  and  $J(2) := \{j, j'\}$ ; let  $K(1, j) := \{k_1, k_2\}$ ,  $K(2, j) := \{k_1\}$ , and  $K(2, j') := \{k'\}$ ; and let  $X(i, j, k) = \{x, y\}$  for all  $i, j, k$ . Now the formula

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

from (2.20) has been given meaning as an actual set. Here is a list of all eight of its elements:

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

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

*Exercise 2.22.* Consider the set

$$B := \prod_{i \in I} \sum_{j \in J(i)} \prod_{k \in K(i, j)} X(i, j, k). \quad (2.23)$$

1. Give the instructions for choosing an element of  $B$  as a nested list, like we did for  $A$  just below (2.20).
2. With  $I, J, K$ , and  $X$  as in Example 2.21, how many elements are in  $B$ ?
3. Write out three of these elements in the style of Example 2.21. ◇

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

### 2.2.2 Expanding products of sums

We will often encounter sums of dependent sets nested within products, as in (2.23). The following proposition helps us work with these; it is sometimes called the *type-theoretic axiom of choice*. Though we will take the time to walk you through its proof, it is almost trivial, once you understand what it's saying. For one thing, it is exactly how you would expect the distributive property of multiplication over addition to work, following the same sort of process that you would use to multiply multi-digit numbers from grade school or polynomials from high school algebra; for another, once the statement is written in Agda, its proof is one short line of Agda code.

**Proposition 2.24** (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

$$\prod_{i \in I} \sum_{j \in J(i)} X(i, j) \cong \sum_{\bar{j} \in \prod_{i \in I} J(i)} \prod_{i \in I} X(i, \bar{j}(i)).^a \quad (2.25)$$

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

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

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

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

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

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

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

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

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

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

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

When  $J(i) = J$  does not depend on  $i \in I$ , the formula in (2.25) becomes much easier.

**Corollary 2.26.** For any set  $I$ , set  $J$ , and sets  $(X(i, j))_{i \in I, j \in J}$ , we have a natural isomorphism

$$\prod_{i \in I} \sum_{j \in J} X(i, j) \cong \sum_{\bar{j}: I \rightarrow J} \prod_{i \in I} X(i, \bar{j}(i)). \quad (2.27)$$

*Proof.* Just take  $J(i) = J$  for all  $i \in I$  in (2.25). Note that dependent functions  $\bar{j}$  in  $\prod_{i \in I} J(i)$  then become standard functions  $\bar{j}: I \rightarrow J$ .  $\square$

It turns out that being able to push  $\prod$  past  $\sum$  as in (2.25) is not a property that is unique to sets. In general, we refer to a category having this property as follows.

**Definition 2.28** (Completely distributive category). A category  $\mathcal{C}$  with all small products and coproducts is *completely distributive* if products distribute over coproducts as in (2.25); that is, for any set  $I$ , sets  $(J(i))_{i \in I}$ , and objects  $(X(i, j))_{i \in I, j \in J(i)}$  from  $\mathcal{C}$ , we have a natural isomorphism

$$\prod_{i \in I} \sum_{j \in J(i)} X(i, j) \cong \sum_{\bar{j} \in \prod_{i \in I} J(i)} \prod_{i \in I} X(i, \bar{j}(i)). \quad (2.29)$$

So Proposition 2.24 states that **Set** is completely distributive. We'll see shortly that **Poly** is, too. Of course, Corollary 2.26 generalizes to all completely distributive categories as well.

**Corollary 2.30.** Let  $\mathcal{C}$  be a completely distributive category. For any set  $I$ , set  $J$ , and objects  $(X(i, j))_{i \in I, j \in J}$  from  $\mathcal{C}$ , we have a natural isomorphism

$$\prod_{i \in I} \sum_{j \in J} X(i, j) \cong \sum_{\bar{j}: I \rightarrow J} \prod_{i \in I} X(i, \bar{j}(i)). \quad (2.31)$$

*Proof.* Again, just take  $J(i) = J$  for all  $i \in I$  in (2.29).  $\square$

**Exercise 2.32.** Let  $\mathcal{C}$  be a completely distributive category. How is the usual distributive law,

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

for  $X, Y, Z \in \mathcal{C}$ , a special case of (2.29)?  $\diamond$

Throughout this book, e.g. in Exercise 2.61, you'll often see an expression consisting of alternating products and sums. Using (2.29), you can always write such an expression as a sum of products, in which every  $\sum$  appears before every  $\prod$  (i.e. in “disjunctive normal form”). This is analogous to how products of sums in high school algebra can always be expanded into sums of products via the distributive property.

**Exercise 2.33.** Let  $I, (J(i))_{i \in I}, (K(i, j))_{(i, j) \in IJ}$  be sets, and for each  $(i, j, k) \in IJK$ , let  $X(i, j, k)$  be an object in a completely distributive category.

1. Rewrite

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

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

2. Rewrite

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

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

3. Rewrite

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

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

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

### 2.2.3 Dependent sums and products of functors $\mathbf{Set} \rightarrow \mathbf{Set}$

Remember: our goal is to define polynomial functors, e.g.  $y^2 + 2y + 1$ , and the maps between them. Since  $y^2$ ,  $y$ , and  $1$  are functors, we just need to interpret sums of

functors  $\mathbf{Set} \rightarrow \mathbf{Set}$ . But we might as well examine products of functors at the same time, because they'll very much come in handy. And both these concepts generalize to limits and colimits in  $\mathbf{Set}^{\mathbf{Set}}$ .

**Proposition 2.34.** The category  $\mathbf{Set}^{\mathbf{Set}}$  has all small limits and colimits, and they are computed pointwise. That is, given a small category  $\mathcal{J}$  and a functor  $F: \mathcal{J} \rightarrow \mathbf{Set}^{\mathbf{Set}}$ , for all  $X \in \mathbf{Set}$ , the limit and colimit of  $F$  satisfy isomorphisms

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

natural in  $X$ .

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

*Example 2.35* (Dependent sums and products of functors  $\mathbf{Set} \rightarrow \mathbf{Set}$ ). For any two functors  $F, G: \mathbf{Set} \rightarrow \mathbf{Set}$ , let

$$(F + G): \mathbf{Set} \rightarrow \mathbf{Set} \quad \text{and} \quad (F \times G): \mathbf{Set} \rightarrow \mathbf{Set}$$

respectively denote the categorical *sum* (i.e. *coproduct*) and *product* of  $F$  and  $G$  in  $\mathbf{Set}^{\mathbf{Set}}$ .<sup>a</sup> Then by Proposition 2.34, for each  $X \in \mathbf{Set}$ ,

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

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

$$\sum_{i \in I} F_i: \mathbf{Set} \rightarrow \mathbf{Set} \quad \text{and} \quad \prod_{i \in I} F_i: \mathbf{Set} \rightarrow \mathbf{Set}$$

respectively denote the categorical *sum* (i.e. *coproduct*) and *product* of the functors  $(F_i)_{i \in I}$ . Then by Proposition 2.34, for each  $X \in \mathbf{Set}$ ,

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

Given a set  $I \in \mathbf{Set}$ , we will also use  $I$  to denote the constant functor that assigns  $I$  to each  $X \in \mathbf{Set}$ . In particular, we denote by  $0, 1: \mathbf{Set} \rightarrow \mathbf{Set}$  the constant functors that respectively assign 0 and 1 to each  $X \in \mathbf{Set}$ . As the set 0 (resp. 1) is the initial (resp. terminal) object in  $\mathbf{Set}$ , Proposition 2.34 tells us that the constant functor 0 (resp. 1) is the initial (resp. terminal) object in  $\mathbf{Set}^{\mathbf{Set}}$ .

<sup>a</sup>We may also denote the product functor  $F \times G$  by  $FG$ .

**Proposition 2.36.** The category  $\mathbf{Set}^{\mathbf{Set}}$  is completely distributive.

*Proof.* This follows directly from the fact that  $\mathbf{Set}$  itself is completely distributive (Proposition 2.24) and the fact that sums and products in  $\mathbf{Set}^{\mathbf{Set}}$  are computed pointwise (Proposition 2.34 and, in particular, Example 2.35).  $\square$

*Exercise 2.37.* 1. Show that for a set  $I \in \mathbf{Set}$  and a functor  $F: \mathbf{Set} \rightarrow \mathbf{Set}$ , the sum of  $I$  copies of  $F$  is isomorphic to the product of the constant functor  $I$  and  $F$ :

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

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

2. So does  $2y$  denote  $2 \times y$  or  $y + y$ , or does it not matter for some reason?  $\diamond$

*Exercise 2.38.* 1. Show that for a set  $I \in \mathbf{Set}$ , the product of  $I$  copies of the identity functor  $y: \mathbf{Set} \rightarrow \mathbf{Set}$  is isomorphic to the functor  $y^I: \mathbf{Set} \rightarrow \mathbf{Set}$  represented by  $I$ :

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

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

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

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

$\diamond$

*Remark 2.39.* From here on out, given  $I \in \mathbf{Set}$  and a functor  $F: \mathbf{Set} \rightarrow \mathbf{Set}$ , we define

$$F^I := \prod_{i \in I} F.$$

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

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

### 2.2.4 What is a polynomial functor?

**Definition 2.40** (Polynomial functor). A *polynomial functor* (or simply a *polynomial*) is a functor  $p: \mathbf{Set} \rightarrow \mathbf{Set}$  such that there exists a set  $I$ , sets  $(p[i])_{i \in I}$ , and an isomorphism

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

to a sum of representables.

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

*Remark 2.41.* Given sets  $I, A \in \mathbf{Set}$ , it follows from Exercise 2.37 that we have an isomorphism of polynomials

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

So when we write down a polynomial, we will often combine identical representable summands  $y^A$  by writing them in the form  $I y^A$ . In particular, the constant functor  $1$  is a representable functor ( $1 \cong y^0$ ), so every constant functor  $I$  is a polynomial functor:  $I \cong \sum_{i \in I} 1$ .

*Example 2.42.* Consider the polynomial  $p := y^2 + 2y + 1$ . It denotes a functor  $\mathbf{Set} \rightarrow \mathbf{Set}$ ; what does this functor do to the set  $X := \{a, b\}$ ? To be very precise and verbose, let's say

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

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

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

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

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

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

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

*Exercise 2.43.* In the verbose style of Example 2.42, write out all the elements of  $p(X)$  for  $p$  and  $X$  as follows:

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  $X := \mathbb{N}$ .

◇

**Proposition 2.44.** Let  $p := \sum_{i \in I} y^{p[i]}$  be an arbitrary polynomial functor. Then  $I \cong p(1)$ , so there is an isomorphism of functors

$$p \cong \sum_{i \in p(1)} y^{p[i]}. \quad (2.45)$$

*Proof.* We need to show that  $I \cong p(1)$ ; the latter claim follows directly. In Exercise 2.17 it was shown that  $I \cong \sum_{i \in I} 1$ , so we just need to show that  $(y^{p[i]})(1) \cong 1$  for every  $i \in I$ . But  $1^{p[i]} \cong 1$  because there is a unique function  $p[i] \rightarrow 1$  for any  $p[i]$ . □

We can draw an analogy between Proposition 2.44 and evaluating  $p(1)$  for a polynomial  $p$  from high school algebra, which yields the sum of the coefficients of  $p$ . The notation in (2.45) will be how we denote arbitrary polynomials from now on.

*Exercise 2.46.* We saw in Proposition 2.44 that for any polynomial  $p$ , e.g.  $p := y^3 + 3y^2 + 4$ , the set  $p(1)$  gives back the set of summands, in this case 8.

What does  $p(0)$  give you?

◇

As a functor  $\mathbf{Set} \rightarrow \mathbf{Set}$ , a polynomial should be able to act not only on sets but also on functions. Below, we explain how.

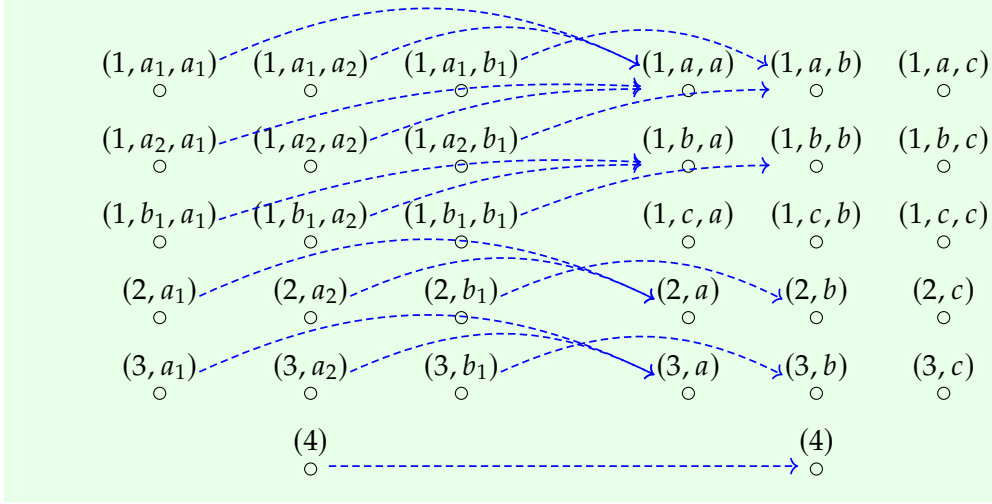
**Proposition 2.47.** Let  $p$  be an arbitrary polynomial functor, which our notation allows us to write it as  $p \cong \sum_{i \in p(1)} y^{p[i]}$ , and let  $f: X \rightarrow Y$  be an arbitrary function. Then  $p(f): p(X) \rightarrow p(Y)$  sends each  $(i, g) \in p(X)$ , with  $i \in p(1)$  and  $g: p[i] \rightarrow X$ , to  $(i, g \circ f)$  in  $p(Y)$ .

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

*Example 2.48.* Suppose  $p := y^2 + 2y + 1$ . Let  $X := \{a_1, a_2, b_1\}$  and  $Y := \{a, b, c\}$ , and let  $f: X \rightarrow Y$  be the function sending  $a_1, a_2 \mapsto a$  and  $b_1 \mapsto b$ . The induced function



$p(f): p(X) \rightarrow p(Y)$ , according to Proposition 2.47, is shown below:



*Exercise 2.49.* Let  $p := y^2 + y$ . Choose a function  $f: 1 \rightarrow 2$  and write out the induced function  $p(f): p(1) \rightarrow p(2)$ .  $\diamond$

## 2.3 Morphisms between polynomial functors

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

**Definition 2.50** (Polynomial morphism, **Poly**). Given polynomial functors  $p$  and  $q$ , a *morphism of polynomial functors* (or a *polynomial morphism*) is a natural transformation  $p \rightarrow q$ . Then **Poly** is the category whose objects are polynomial functors and whose morphisms are polynomial morphisms.

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

### 2.3.1 Coproducts of polynomials

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

**Proposition 2.51.** The category **Poly** has arbitrary coproducts, coinciding with coproducts in  $\mathbf{Set}^{\mathbf{Set}}$  given by the operation  $\sum_{i \in I}$ .

*Proof.* By Example 2.35, the category  $\mathbf{Set}^{\mathbf{Set}}$  has arbitrary coproducts given by  $\sum_{i \in I}$ . The full subcategory inclusion  $\mathbf{Poly} \rightarrow \mathbf{Set}^{\mathbf{Set}}$  reflects these coproducts, and by definition  $\mathbf{Poly}$  is closed under the operation  $\sum_{i \in I}$ .  $\square$

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

$$\sum_{i \in I} p_i \cong \sum_{i \in I} \sum_{j \in p_i(1)} y^{p_i[j]} \cong \sum_{(i,j) \in \sum_{i \in I} p_i(1)} y^{p_i[j]}, \quad (2.52)$$

which coincides with our notion of sums of functors  $\mathbf{Set} \rightarrow \mathbf{Set}$  from Example 2.35. Of course, it also coincides with our notion of polynomial addition from high school algebra—just add all the terms together! Binary coproducts are given by binary sums of functors, appropriately denoted by  $+$ , while the initial object of  $\mathbf{Poly}$  is the constant polynomial 0.

In particular, (2.52) implies that for any polynomials  $p$  and  $q$ , their coproduct  $p + q$  is given as follows. The position-set of  $p + q$  is the coproduct of sets  $p(1) + q(1)$ . At position  $(1, i) \in p(1) + q(1)$  with  $i \in p(1)$ , the directions of  $p + q$  are just the  $p[i]$ -directions; at position  $(2, j) \in p(1) + q(1)$  with  $j \in q(1)$ , the directions of  $p + q$  are just the  $q[j]$ -directions.

### 2.3.2 Polynomial morphisms, concretely

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

*Exercise 2.53.* Given a set  $S$  and a polynomial  $q$ , show that a polynomial morphism  $y^S \rightarrow q$  can be identified with an element of the set  $q(S)$ . That is, there is an isomorphism

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

Moreover, show that this isomorphism is natural in both  $S$  and  $q$ . Hint: Use the Yoneda lemma (Lemma 2.11).  $\diamond$

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

**Proposition 2.54.** Let  $p := \sum_{i \in p(1)} y^{p[i]}$  and  $q := \sum_{j \in q(1)} y^{q[j]}$  be polynomials. Then we have an isomorphism

$$\mathbf{Poly}(p, q) \cong \prod_{i \in p(1)} \sum_{j \in q(1)} p[i]^{q[j]} \quad (2.55)$$

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

$$\begin{array}{ccc} p(1) & \xrightarrow{f_1} & q(1) \\ & \searrow f^\# \swarrow & \\ p[-] & \xleftarrow{\quad} & q[-] \end{array} \quad \text{Set} \quad (2.56)$$

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

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

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

so applying Exercise 2.53 (i.e. the Yoneda lemma) and unraveling the definitions of  $p$  and  $q$  yields (2.55).

The expression on the right hand side of (2.55) is the set of dependent functions  $f: (i \in p(1)) \rightarrow \sum_{j \in q(1)} p[i]^{q[j]}$ , each of which is uniquely determined by its components:  $\pi_1 \circ f$ , which sends  $i \in p(1)$  to  $\pi_1(f(i)) \in q(1)$ , and  $\pi_2 \circ f$ , which sends  $i \in p(1)$  with  $j := \pi_1(f(i))$  to an element of  $p[i]^{q[j]}$ , i.e. a function  $q[j] \rightarrow p[i]$ . These can be identified respectively with a (non-dependent) function  $f_1 := \pi_1 \circ f$  from  $p(1) \rightarrow q(1)$  and a natural transformation  $f^\#: q[f_1(-)] \rightarrow p[-]$ .  $\square$

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

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

When we wish to emphasize the arena perspective, we will call the data of  $(f_1, f^\#)$  an *arena morphism*, a *dependent lens*, or simply a *lens* between  $p$  and  $q$  (see Section 2.3.3); but since Proposition 2.54 tells us that polynomial morphisms and arena morphisms carry the same data, we may use these terms interchangeably. In fact, in future chapters we will primarily refer to the morphisms of **Poly** as *lenses*. If you're familiar with lens terminology from a functional programming context, we'll see in Section 2.3.3 how our notion of dependent lenses is related.

This forward on-positions/backward on-directions formulation may still seem a little complicated, so here is some decision-making intuition. Recall from Section 1.2 that we may view an arena as a scenario, each position of an arena as a menu, and the directions at that position as the options available on that menu. Every time we select an option off a menu, we are making a decision. Then the morphisms  $f: p \rightarrow q$  are the ways to *delegate*  $p$ 's decisions to  $q$ .

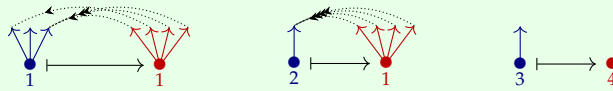
You should think of this as an agent Pat in scenario  $p$  needing to make a decision, so they ask a second agent Quinn in scenario  $q$  to make another decision that will then inform the decision that Pat will make. Each one of Pat's menus in scenario  $p$ , say  $i \in p(1)$ , is passed forward ("delegated") to a menu in scenario  $q$ , say  $j \in q(1)$ , for Quinn to choose an option from. Then each option  $b \in q[j]$  that Quinn could select from menu  $j$  is passed back to Pat as an option  $a \in p[i]$  on  $p$ 's original menu  $i$ . The morphism tells us how each menu in  $p$  is assigned a menu in  $q$  to watch, as well as how each choice of option from  $q$ 's menu dictates a choice of option from  $p$ 's menu. Henceforth, we will often conflate the agent making a decision with the scenario that they are in, writing statements like " $p$  chooses an option from its menu."

*Example 2.57.* Let  $p := y^3 + 2y$  and  $q := y^4 + y^2 + 2$ . Here they are, depicted as corolla forests:



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

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



This represents one morphism  $p \rightarrow q$ .

So how many different morphisms are there from  $p$  to  $q$ ? The first  $p$ -position can be sent to any  $q$ -position: 1, 2, 3, or 4. Sending it to 1 requires choosing how each of the four options ( $q[1] = 4$ ) are to be assigned one of  $p[1] = 3$  options; there are  $3^4$  ways to

do this. Similarly, we can calculate the remaining ways to handle the first  $p$ -position, then add them up: there are  $3^4 + 3^2 + 3^0 + 3^0 = 92$  ways total.

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

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

Unsurprisingly, this is exactly what is given by (2.55):

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

- Exercise 2.58.*
1. Draw the corolla forests associated to  $p := y^3 + y + 1$ ,  $q := y^2 + y^2 + 2$ , and  $r := y^3$ .
  2. Give an example of a morphism  $p \rightarrow q$  and draw it as we did in Example 2.57.
  3. Explain your morphism intuitively as a delegation of decisions.
  4. Explain in those terms why there can't be any morphisms  $p \rightarrow r$ . ◇

*Exercise 2.59.* For any polynomial  $p$  and set  $A$ , e.g.  $A = 2$ , the Yoneda lemma gives an isomorphism  $p(A) \cong \mathbf{Poly}(y^A, p)$ .

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

*Exercise 2.60.* For each of the following polynomials  $p, q$ , compute the number of morphisms  $p \rightarrow q$ .

1.  $p = y^3$ ,  $q = y^4$ .
2.  $p = y^3 + 1$ ,  $q = y^4$ .
3.  $p = y^3 + 1$ ,  $q = y^4 + 1$ .
4.  $p = 4y^3 + 3y^2 + y$ ,  $q = y$ .
5.  $p = 4y^3$ ,  $q = 3y$ . ◇

*Exercise 2.61.* 1. Show that the following are isomorphic:

$$\mathbf{Poly}(p, q) \cong \prod_{i \in p(1)} \sum_{j \in q(1)} \prod_{b \in q[j]} \sum_{a \in p[i]} 1 \quad (2.62)$$

2. Show that the following are isomorphic:

$$\mathbf{Poly}(p, q) \cong? \sum_{f_1: p(1) \rightarrow q(1)} \prod_{j \in q(1)} \mathbf{Set}\left(q[j], \prod_{\substack{i \in p(1), \\ f_1(i)=j}} p[i]\right) \quad (2.63)$$

3. Describe in the language of decision-making how any element of the right-hand side of (2.63) gives a way of delegating decisions from  $p$  to  $q$ .  $\diamond$

*Exercise 2.64.* Use (2.52) and (2.55) to verify that

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

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

*Example 2.65 (Derivatives).* The *derivative* of a polynomial  $p$ , denoted  $\dot{p}$ , is defined as follows:

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

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

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

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

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

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

This natural transformation comes up in computer science in the context of “plugging in to one-hole contexts”; we will not explore that here, but see [McB01] and [Abb+03] for more info.

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

*Exercise 2.66.* Show that  $\dot{p}(1)$  is isomorphic to the set of all directions of  $p$  (i.e. the union of all direction-sets of  $p$ ), so there is a canonical function  $\pi_p: \dot{p}(1) \rightarrow p(1)$  that sends each direction  $a$  of  $p$  to the position  $i$  of  $p$  for which  $a \in p[i]$ .  $\diamond$

*Exercise 2.67.* The derivative is not very well-behaved category-theoretically. However, it is intriguing. Below  $p, q \in \mathbf{Poly}$ .

1. Explain the canonical map  $\dot{p}y \rightarrow p$  from Example 2.65 in more detail.
2. Is there always a canonical map  $p \rightarrow \dot{p}$ ?
3. Is there always a canonical map  $\dot{p} \rightarrow p$ ?
4. If given a map  $p \rightarrow q$ , does one get a map  $\dot{p} \rightarrow \dot{q}$ ?
5. We will define the binary operations  $\otimes$  and  $[-, -]$  on  $\mathbf{Poly}$  later on in (2.90) and (3.78), and in Exercise 3.83, you will be able to use Exercise 3.81 to deduce that

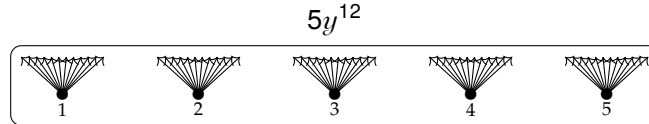
$$[p, y] \otimes p \cong \sum_{f \in \prod_{i \in p(1)} p[i]} \sum_{i \in p(1)} y^{p(1) \times p[i]}, \quad (2.68)$$

Is there always a canonical map  $[p, y] \otimes p \rightarrow \dot{p}$ ?

6. When talking to someone who explains maps  $p \rightarrow \dot{q}$  in terms of “unassigned directions,” how might you describe what is modeled by a map  $py \rightarrow q$ ?  $\diamond$

### 2.3.3 Arena morphisms as dependent lenses

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



The formula for morphisms between these is particularly simple:

$$\begin{aligned} \mathbf{Poly} \left( B_1 y^{A_1}, B_2 y^{A_2} \right) &\cong \prod_{b \in B_1} \sum_{b' \in B_2} A_1^{A_2} \\ &\cong \mathbf{Set}(B_1, B_2 \times A_1^{A_2}) \\ &\cong \mathbf{Set}(B_1, B_2) \times \mathbf{Set}(B_1 \times A_2, A_1). \end{aligned} \quad (2.55)$$

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

$$\mathbf{Poly} \left( B_1 y^{A_1}, B_2 y^{A_2} \right) \cong \left\{ (f_1, f^\#) \left| \begin{array}{l} f_1: B_1 \rightarrow B_2 \\ f^\#: B_1 \times A_2 \rightarrow A_1 \end{array} \right. \right\}$$

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

The monomials in **Poly** and the morphisms between them form a full subcategory of **Poly**, and it has been called *the category of bimorphic lenses* [Hed18]. It comes up in functional programming. The functions  $f_1, f^\#$  corresponding to a morphism  $f: B_1 y^{A_1} \rightarrow B_2 y^{A_2}$  are given special names:

$$\begin{aligned} \text{get} &:= f_1: B_1 \rightarrow B_2 \\ \text{put} &:= f^\#: B_1 \times A_2 \rightarrow A_1 \end{aligned} \tag{2.69}$$

The idea is that each position  $b \in B_1$  of  $B_1 y^{A_1}$  “gets” a position  $f_1(b) \in B_2$  of  $B_2 y^{A_2}$ , and given  $b \in B_1$ , every direction at  $f_1(b)$  in  $A_2$  “puts” a direction back at  $b$  in  $A_1$ .

So an arena morphism between two monomials is just a bimorphic lens. Then an arena morphism between any two arenas is a more generalized lens: a *dependent lens*, where the direction-sets depend on the positions. This is why we will henceforth refer to arena morphisms simply as *lenses*.

*Example 2.70.* Consider the monomial  $Sy^S$ . As an arena, its position-set is  $S$ , and its direction-set at each position  $s \in S$  is again just  $S$ . In the language of decision-making, each  $s \in S$  is a menu whose options are themselves just the menus in  $S$  again. Notice that there is a natural way to string together a series of such decisions into a cycle: at each step, you start at some element of  $S$ , and the option you select is the element of  $S$  that you will move to next. We will start to formalize this idea in Example 3.43 and continue this work throughout the following chapters.

A lens  $(\text{get}, \text{put}): Sy^S \rightarrow Ty^T$  is as usual a way to delegate decisions of  $Sy^S$  to decisions of  $Ty^T$ . When you need to select an option from the menu  $s \in S$ , you ask your friend to check the menu  $\text{get}(s) \in T$  for help. If your friend selects option  $t \in T$ , then you know to select option  $\text{put}(s, t) \in S$ . Then the options you have each selected are your new positions: your friend moves to  $t$ , you move to  $\text{put}(s, t)$ , and you may each check the corresponding menus at your new positions.

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

1. If your friend chooses to stay put, then you should stay put, too. This is reflected by the equation

$$\text{put}(s, \text{get}(s)) = s.$$

2. After your friend moves, and you move accordingly, you should delegate the decision at your new location to the decision at your friend’s new location. This



is reflected by the equation

$$\text{get}(\text{put}(s, t)) = t.$$

3. If your friend moves to  $t$ , then to  $t'$ , the place where you end up should be where you would have ended up if your friend had moved directly to  $t'$  in the first place. This is reflected by the equation

$$\text{put}(\text{put}(s, t), t') = \text{put}(s, t')$$

Such a lens is known to functional programmers as a *very well-behaved lens*; the three conditions above are its *lens laws*. But we will see these conditions emerge from more general theory in Example 6.84.

### 2.3.4 Translating between natural transformations and lenses

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

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

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

Our Rosetta Stone turns out to be the proof of the Yoneda lemma. The lemma itself forms the crux of the proof of Proposition 2.54, that these two formulations of polynomial morphisms are equivalent; so unraveling this proof reveals the translation we seek.

**Proposition 2.71.** Let  $p$  and  $q$  be polynomial functors, and let  $(f_1, f^\sharp)$  be a morphism between their associated arenas. Then the isomorphism in (2.55) sends  $(f_1, f^\sharp)$  to the natural transformation  $f: p \rightarrow q$  whose  $X$ -component  $f_X: p(X) \rightarrow q(X)$  for each  $X \in \mathbf{Set}$  sends every

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

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

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

*Proof.* As an element of the product over  $I$  on the right hand side of (2.55), the pair  $(f_1, f^\sharp)$  can equivalently be thought of as multiple pairs  $((f_1(i), f_i^\sharp))_{i \in I}$ . Fixing  $i \in I$ , the

pair  $(f_1(i), f_i^\#)$  is an element of

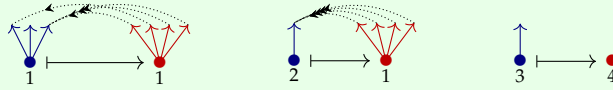
$$\sum_{j \in q(1)} p[i]^{q[j]} = q(p[i])$$

(so  $f_1(i) \in q(1)$  and  $f_i^\#: q[f_1(i)] \rightarrow p[i]$ ). By the Yoneda lemma (Lemma 2.11), 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  $(f_1(i), f_i^\#)$  to the natural transformation  $f^i: y^{p[i]} \rightarrow q$  whose  $X$ -component is the function  $f_X^i: X^{p[i]} \rightarrow q(X)$  given by sending  $g: p[i] \rightarrow X$  to

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

Taken together, the natural transformations  $(f^i)_{i \in I}$  form an element of  $\prod_{i \in I} \mathbf{Poly}(y^{p[i]}, q)$ . Applying the universal property of coproducts, as in the proof of Proposition 2.54, we find that  $(f^i)_{i \in I}$  corresponds to the natural transformation  $f: p \rightarrow q$  we desire.  $\square$

*Example 2.72.* Let us return to the polynomials  $p := y^3 + 2y$  and  $q := y^4 + y^2 + 2$  from Example 2.57 and the morphism  $f: p \rightarrow q$  depicted below:



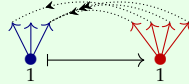
Fix a set  $X := \{a, b, c, d, e\}$ . When viewed as a natural transformation, the morphism  $f$  has as its  $X$ -component a function  $f_X: p(X) \rightarrow q(X)$ . In other words, for any element of  $p(X)$ , the morphism  $f$  should be able to give us an element of  $q(X)$ .

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



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

Proposition 2.71 tells us that  $f_X(1, g)$  can be read off of the forest depiction of  $f$  at position 1:



To draw  $f_X(1, g)$ , we first draw the  $q$ -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 (as given by  $f_i^\sharp$ ) to a  $p[1]$ -leaf, and use the label there that is given by  $(1, g)$ . So  $f_X(1, g)$  looks like



Proposition 2.71 lets us translate from lenses to natural transformations. The following corollary tells us how to go in the other direction. In particular, it justifies the notation  $f_1$  for the on-positions function of  $f$ .

**Corollary 2.73.** Let  $p$  and  $q$  be polynomial functors, and let  $f: p \rightarrow q$  be a natural transformation between them. Then the isomorphism in (2.55) sends  $f$  to the lens  $(f_1, f^\sharp)$  for which  $f_1: p(1) \rightarrow q(1)$  is the 1-component of  $f$  and, for each  $i \in p(1)$ , we have

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

*Proof.* By Proposition 2.71, the 1-component of  $f$  is a function  $p(1) \rightarrow q(1)$  sending every  $i \in p(1)$  to  $f_1(i) \in q(1)$ , so the on-positions function  $f_1$  is indeed equal to the 1-component of  $f$ . Also, the  $p[i]$ -component  $f_{p[i]}: p(p[i]) \rightarrow q(p[i])$  sends every  $(i, \text{id}_{p[i]}) \in p(p[i])$ , with  $i \in p(1)$ , to  $(f_1(i), f_i^\sharp \circ \text{id}_{p[i]}) = (f_1(i), f_i^\sharp)$ .  $\square$

### 2.3.5 Identity and composition of lenses

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

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

*Exercise 2.74 (Identity lenses).* Let  $p$  be a polynomial and let  $\text{id}_p: p \rightarrow p$  be its identity natural transformation, whose  $X$ -component  $(\text{id}_p)_X: p(X) \rightarrow p(X)$  for each  $X \in \mathbf{Set}$  is the identity function on  $p(X)$ ; that is,  $(\text{id}_p)_X = \text{id}_{p(X)}$ .

Use Corollary 2.73 to show that the lens  $((\text{id}_p)_1, (\text{id}_p)^\sharp)$  associated to  $\text{id}_p$  is such that  $(\text{id}_p)_1: p(1) \rightarrow p(1)$  and  $(\text{id}_p)_i^\sharp: p[(\text{id}_p)_1(i)] \rightarrow p[i]$  for  $i \in p(1)$  are all identity functions.  $\diamond$

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

*Exercise 2.75 (Composing lenses).* Let  $p, q$ , and  $r$  be polynomials, let  $f: p \rightarrow q$  and  $g: q \rightarrow r$  be natural transformations, and let  $h := f \circ g$  be their composite, whose  $X$ -component  $h_X: p(X) \rightarrow r(X)$  for each  $X \in \mathbf{Set}$  is the composite of the  $X$ -components

of  $f$  and  $g$ ; that is,  $h_X = f_X \circ g_X$ .

Use Corollary 2.73 to show that the lens  $(h_1, h^\#)$  associated to  $h$  satisfies  $h_1 = f_1 \circ g_1$  and  $h_i^\# = g_{f_1(i)}^\# \circ f_i^\#$  for all  $i \in p(1)$ .  $\diamond$

*Example 2.76* (Commutative diagrams in **Poly**). The above exercise tells us how to interpret commutative diagrams in **Poly** as commutative diagrams in the more familiar setting of **Set**. Given polynomials  $p, q, r$  and natural transformations  $f: p \rightarrow q, g: q \rightarrow r$ , and  $h: p \rightarrow r$ , the diagram

$$\begin{array}{ccc} p & \xrightarrow{f} & q \\ & \searrow h & \downarrow g \\ & & r \end{array}$$

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

$$\begin{array}{ccc} p(1) & \xrightarrow{f_1} & q(1) \\ & \searrow h_1 & \downarrow g_1 \\ & & r(1) \end{array}$$

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

$$\begin{array}{ccc} p[i] & \xleftarrow{f_i^\#} & q[f_1(i)] \\ & \nwarrow h_i^\# & \uparrow g_{f_1(i)}^\# \\ & & r[h_1(i)] \end{array}$$

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

*Exercise 2.77.* Verify that, for  $p, q \in \mathbf{Poly}$ , the polynomial  $p + q$  given by the binary sum of  $p$  and  $q$  satisfies the universal property of the coproduct of  $p$  and  $q$ . That is, provide morphisms  $\iota: p \rightarrow p + q$  and  $\kappa: q \rightarrow p + q$ , then show that for any other polynomial  $r$  with morphisms  $f: p \rightarrow r$  and  $g: q \rightarrow r$ , there exists a unique morphism  $h: p + q \rightarrow r$ —shown dashed—making the following diagram commute:

$$\begin{array}{ccccc} p & \xrightarrow{\iota} & p + q & \xleftarrow{\kappa} & q \\ & \searrow f & \downarrow \text{dashed } h & \swarrow g & \\ & & r & & \end{array} \quad (2.78)$$

Hint: Use Example 2.76 to determine whether a diagram commutes.  $\diamond$

*Exercise 2.79* (A functor **Top**  $\rightarrow$  **Poly**). This exercise is for those who know what topological spaces and continuous maps are. It will not be used again in this book.

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

◇

## 2.4 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. But there is an added bonus about the monoidal products on **Poly** that we'll introduce that isn't shared by some fancy monoidal products on other categories like "the tensor product of modules over a commutative ring": if you know how to add, multiply, and exponentiate expressions from high school algebra, then you already know how to compute these monoidal products!

We have already seen one monoidal structure on **Poly**: the cocartesian monoidal structure, which gives **Poly** its finite coproducts. In fact, we know from Proposition 2.51 that **Poly** has all coproducts—and they are given by an operation that looks just like addition. We'll quickly see in Section 2.4.1 that **Poly** has all products as well, giving it a cartesian monoidal structure. Can you guess what that operation looks like?

Much of Part II will focus on the astonishing features of another monoidal structure, an asymmetric one, which we will hold off on defining—we'll save its surprises for when we can better savor them. But here in Section 2.4.2, we will introduce a third symmetric monoidal structure, given by an operation you probably weren't allowed to do to polynomials back in high school. This monoidal structure will really come in handy when we apply **Poly** to dynamics in the next chapter.

### 2.4.1 The categorical product

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

**Proposition 2.80.** 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 2.51.

By Example 2.35, the category  $\mathbf{Set}^{\mathbf{Set}}$  has arbitrary products given by  $\prod_{i \in I}$ . The full subcategory inclusion  $\mathbf{Poly} \rightarrow \mathbf{Set}^{\mathbf{Set}}$  reflects these products. It remains to show that  $\mathbf{Poly}$  is closed under the operation  $\prod_{i \in I}$ .

By Proposition 2.36,  $\mathbf{Set}^{\mathbf{Set}}$  is completely distributive. Hence, given polynomials  $(p_i)_{i \in I}$ , we can use (2.29) to write their product in  $\mathbf{Set}^{\mathbf{Set}}$  as

$$\prod_{i \in I} p_i \cong \prod_{i \in I} \sum_{j \in p_i(1)} y^{p_i[j]} \cong \sum_{\tilde{j} \in \prod_{i \in I} p_i(1)} \prod_{i \in I} y^{p_i[\tilde{j}(i)]} \cong \sum_{\tilde{j} \in \prod_{i \in I} p_i(1)} y^{\sum_{i \in I} p_i[\tilde{j}(i)]}, \quad (2.81)$$

which, as a coproduct of representables, is in  $\mathbf{Poly}$ .  $\square$

**Corollary 2.82.** The category  $\mathbf{Poly}$  is completely distributive.

*Proof.* This is a direct consequence of the fact that  $\mathbf{Poly}$  has arbitrary (co)products coinciding with (co)products in  $\mathbf{Set}^{\mathbf{Set}}$  (Propositions 2.51 and 2.80) and the fact that  $\mathbf{Set}^{\mathbf{Set}}$  itself is completely distributive (Proposition 2.36).  $\square$

The result above will allow us to apply (2.29), or sometimes specifically (2.31), to push  $\prod$ 's past  $\sum$ 's of polynomials whenever we so desire.

*Exercise 2.83.* Use (2.55) 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.  $\diamond$

*Exercise 2.84.* Let  $p_1 := y + 1$ ,  $p_2 := y + 2$ , and  $p_3 := y^2$ . What is  $\prod_{i \in 3} p_i$  according to (2.81)? Is the answer what you would expect?  $\diamond$

It follows from (2.81) that the terminal object of  $\mathbf{Poly}$  is 1, and that binary products are given by

$$p \times q \cong \sum_{i \in p(1)} \sum_{j \in q(1)} y^{p[i] + q[j]}. \quad (2.85)$$

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

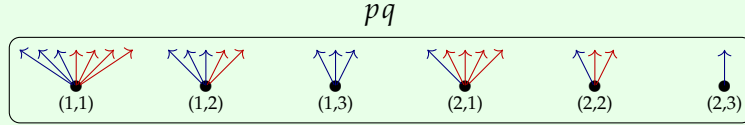
$$pq := p \times q \quad (\text{Notation})$$

*Example 2.86.* We can draw the product of two polynomials in terms of their associated

forests. Let  $p := y^3 + y$  and  $q := y^4 + y^2 + 1$ .



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



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

*Exercise 2.87.* 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  $\pi: pq \rightarrow p$  and  $\varphi: pq \rightarrow q$  behave as you might expect: on positions, they are the projections from  $(pq)(1) \cong p(1) \times q(1)$  to  $p(1)$  and  $q(1)$ , respectively; on directions, they are the inclusions  $p[i] \rightarrow p[i] + q[j]$  and  $q[j] \rightarrow p[i] + q[j]$  for each position  $(i, j)$  of  $pq$ .

*Exercise 2.88.* Verify that, for  $p, q \in \mathbf{Poly}$ , the polynomial  $pq$  given by (2.85) along with the maps  $\pi: pq \rightarrow p$  and  $\varphi: pq \rightarrow q$  described above satisfy the universal property of the product of  $p$  and  $q$ .

◇

### 2.4.2 The parallel product

There is a closely related monoidal structure on  $\mathbf{Poly}$  that will be useful for putting dynamical systems in parallel.

**Definition 2.89** (Parallel product of polynomials). Let  $p$  and  $q$  be polynomials. Their *parallel product*, denoted  $p \otimes q$ , is given by the formula:

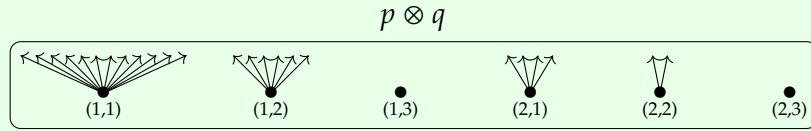
$$p \otimes q \cong \sum_{i \in p(1)} \sum_{j \in q(1)} y^{p[i] \times q[j]}. \quad (2.90)$$

One should compare this with the formula for the product of polynomials shown in (2.85). The difference is that the parallel product multiplies exponents where the categorical product adds them.

*Example 2.91.* We can draw the parallel product of two polynomials in terms of their associated forests. Let  $p := y^3 + y$  and  $q := y^4 + y^2 + 1$ .



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



*Exercise 2.92.* 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 2.93.* Let  $p := y^2 + y$  and  $q := 2y^4$ .

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

◇



*Exercise 2.94.* Consider the polynomials  $p := 2y^2 + 3y$  and  $q := y^4 + 3y^3$ .

1. What is  $p \times q$ ?
2. What is  $p \otimes q$ ?
3. What is the product of the following purely formal expression we'll see *only this once!*:

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

The factors of the above product are called Dirichlet series.

4. Describe the connection between the last two parts. (An alternative name we give for the parallel product  $\otimes$  is the *Dirichlet product*.)  $\diamond$

*Exercise 2.95.* What is  $(3y^5 + 6y^2) \otimes 4$ ? Hint:  $4 = 4y^0$ .  $\diamond$

*Exercise 2.96.* 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  $p \otimes q \cong q \otimes p$ .  $\diamond$

In Exercise 2.96, we have gone most of the way to proving that  $(\mathbf{Poly}, y, \otimes)$  is a symmetric monoidal category.

**Proposition 2.97.** The category  $\mathbf{Poly}$  has a symmetric monoidal structure  $(y, \otimes)$  where  $\otimes$  is the parallel product from Definition 2.89.

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

$$(f \otimes g)_1(i, j) := (f_1(i), g_1(j))$$

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

$$(f \otimes g)_{(i,j)}^\#(a, b) := (f_i^\#(a), g_j^\#(b)).$$

Then Exercise 2.96 gives us the unitors, associator, and braiding. We have not proven the functoriality of  $\otimes$ , the naturality of the isomorphisms from Exercise 2.96, or all the coherences between these isomorphisms, but we ask the reader to take them on trust or to check them for themselves. Alternatively, we may invoke the Day convolution to obtain the monoidal structure  $(y, \otimes)$  directly (see Proposition 2.102).  $\square$

*Exercise 2.98.* 1. If  $p = A$  and  $q = B$  are constant polynomials, what is  $p \otimes q$ ?  
 2. If  $p = A$  is constant and  $q$  is arbitrary, what can you say about  $p \otimes q$ ?  
 3. If  $p = Ay$  and  $q = By$  are linear polynomials, what is  $p \otimes q$ ?  
 4. For arbitrary  $p, q \in \mathbf{Poly}$ , what is the relationship between the sets  $(p \otimes q)(1)$  and  $p(1) \times q(1)$ ?  $\diamond$

*Exercise 2.99.* Which of the following classes of polynomials are closed under  $\otimes$ ? Note also whether they contain  $y$ .

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

*Exercise 2.100.* What is the smallest class of polynomials that's closed under  $\otimes$  and contains  $y$ ?  $\diamond$

*Exercise 2.101.* Show that for any  $p_1, p_2, q \in \mathbf{Poly}$  there is an isomorphism

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

**Proposition 2.102.** For any monoidal structure  $(I, \star)$  on  $\mathbf{Set}$ , there is a corresponding monoidal structure  $(y^I, \odot)$  on  $\mathbf{Poly}$ , where  $\odot$  is the Day convolution. Moreover,  $\odot$  distributes over coproducts.

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

*Proof.* Any monoidal structure  $(I, \odot)$  on  $\mathbf{Set}$  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  $\mathbf{Poly}$ , it suffices to show that  $\mathbf{Poly}$  is closed under the Day convolution.

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

$$p \odot q \cong \int^{(A,B) \in \mathbf{Set}^2} y^{A \star B} \times p(A) \times q(B). \quad (2.103)$$

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

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

So because products distribute over coproducts in **Set** and coends commute with coproducts, we can rewrite (2.103) as

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

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

$$p \odot q \cong \sum_{(i,j) \in p(1) \times q(1)} y^{p[i] \star q[j]} \quad (2.104)$$

in **Poly**. That the Day convolution distributes over coproducts also follows from the fact that products distribute over coproducts in **Set** and coends commute with coproducts; or, alternatively, directly from (2.104).

We observe that (2.104) gives  $(y^I, \odot) = (1, \times)$  when  $(I, \star) = (0, +)$  and  $(y^I, \odot) = (y, \otimes)$  when  $(I, \star) = (1, \times)$ .  $\square$

*Exercise 2.105.* 1. Show that the operation  $(A, B) \mapsto A + AB + B$  on **Set** is associative.  
 2. Show that 0 is unital for the above operation.  
 3. Let  $(1, \odot)$  denote the corresponding monoidal structure on **Poly**. Compute the monoidal product  $(y^3 + y) \odot (2y^2 + 2)$ .  $\diamond$

*Remark 2.106.* Monoids in **Poly** with respect to the parallel product  $\otimes$  are particularly interesting—they have a kind of collective semantics, letting agents aggregate their contributions and distribute returns on those contributions in a coherent way. We leave discussion of them to future work, so as not to distract us from our main story.

## 2.5 Summary and further reading

In this chapter we explained the mathematics behind our main object of study in this book, the category **Poly** of polynomial functors. A polynomial  $p = \sum_{i \in I} y^{p[i]}$  can be considered as

1. combinatorial data: an indexed family of sets  $(p[i])_{i \in I}$ ;
2. a picture: for each  $i \in I$ , a corolla with  $p[i]$ -many leaves;
3. a functor **Set**  $\rightarrow$  **Set**: for each  $X : \mathbf{Set}$ , a new set  $\sum_{i \in I} X^{p[i]}$ .

The last of these ties anchors us to the rest of category theory, so that rather than needing to ask for morphisms between combinatorial objects or between pictures, we can ask for morphisms between functors **Set**  $\rightarrow$  **Set**, and that's well established

territory. Once we said that maps  $p \rightarrow q$  between polynomial functors should just be natural transformations, we translated that back to the combinatorial and pictorial settings and saw the “on-positions, on-directions” description. We want to emphasize that we will be referring to each morphism (natural transformation)  $p \rightarrow q$  between polynomials as a *lens*.

Once **Poly** was defined, we considered various properties it has, e.g. that it has all products and coproducts, and that these distribute:  $\prod \Sigma \rightarrow \Sigma \prod$ .

$$\begin{aligned} \sum_{a \in A} p_a &:= \sum_{(a,i) \in \sum_{a \in A} p_a(1)} y^{p_a[i]} & \prod_{a \in A} p_a &:= \sum_{i \in \prod_{a \in A} p_a(1)} y^{\sum_{a \in A} p_a[ia]} \\ p_1 + p_2 &:= \sum_{(a,i) \in \{(1,i) | i \in p_1\} + \{(2,i) | i \in p_2\}} y^{p_a[i]} & p_1 \times p_2 &:= \sum_{(i_1, i_2) \in p_1(1) \times p_2(1)} y^{p_1[i_1] + p_2[i_2]} p \end{aligned}$$

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

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

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**:

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

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 can be found in [GK12]; 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/>.

# Dynamical systems as dependent lenses

Let's start putting all this Poly stuff to use.

## 3.1 Moore machines

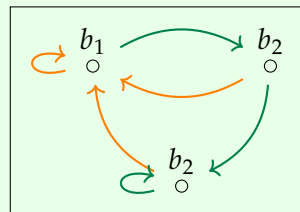
We begin with our simplest example of a dynamical system: a deterministic machine with internal states that can return output and be updated according to input.

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

$$\begin{aligned} \text{return}: S &\rightarrow B \\ \text{update}: S \times A &\rightarrow S \end{aligned}$$

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

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



(3.3)

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

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

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

...

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

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

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

for all  $k \in \mathbb{N}$ . We'll see that **Poly** gives us a more concise way to express this in Example 7.52.

### 3.1.1 Moore machines as lenses

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

$$(\text{get}, \text{put}): Sy^S \rightarrow By^A,$$

with  $\text{get} := \text{return}$  and  $\text{put} := \text{update}$ .<sup>1</sup> Given such a Moore machine, we will call the monomial  $By^A$  the *interface*, because it encodes how an outsider can interact with the

<sup>1</sup>Recall from (2.69) that “get” and “put” are our special names for the on-positions and on-directions functions of a lens between monomials.

machine, namely the possible inputs and outputs; and we will call the monomial  $Sy^S$  the *state system*.

*Exercise 3.4.* Write the Moore machine from Example 3.2 as a lens between monomials.

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

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

3. What is the on-positions function “get”?
4. What is the on-directions function “put”?

◇

### 3.1.2 More Moore machines

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

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

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

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

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

*Exercise 3.7.* Explain in words what the Moore machine in Example 3.6 does.

◇

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

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

If the machine begins in state  $b_0$  and is given a sequence  $(a_1, a_2, \dots)$  of elements in  $A$ , the machine’s outputs will be  $(b_0, f(a_1), f(a_2), \dots)$ . We could say this machine is

*memoryless*, because at no point does the state of the machine depend on any previous states.

*Exercise 3.9.* Suppose we have a function  $f: A \times B \rightarrow B$ .

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

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

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

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

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

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

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

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



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

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

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

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

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

### Deterministic state automata

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

**Definition 3.14** (Deterministic state automaton, language). A *deterministic state automaton* consists of

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

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

$$s_{k+1} := u(s_k, a_{k+1})$$

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

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

*Remark 3.15.* When we study a deterministic state automaton, we are usually interested in which words the automaton accepts and, more generally, what language the automaton recognizes. While intuitive, the condition we provided for when an automaton accepts a word is rather cumbersome to work with. In Example 7.51, we'll see a simpler way of saying whether an automaton accepts a word, as well as specifying what language the automaton recognizes. Better yet, we'll find that this alternative formulation arises naturally from the theory of **Poly**.

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

$$y \rightarrow Sy^S \rightarrow 2y^A.$$

*Proof.* A map  $y \rightarrow Sy^S$  can be identified with an initial state  $s_0 \in S$ . A map  $Sy^S \rightarrow 2y^A$  consists of a function  $f: S \rightarrow 2$ , which can be identified with a subset of accept states  $F \subseteq S$ , together with an update function  $u: S \times A \rightarrow S$ .  $\square$

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

## 3.2 Dependent dynamical systems

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

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

$$\varphi: Sy^S \rightarrow p$$

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

The lens’s on-positions function  $\varphi_1: S \rightarrow p(1)$  is called the *return function*, and for each  $s \in S$ , the lens’s on-directions function  $\varphi_s^\# : p[\varphi_1(s)] \rightarrow S$  is called the *update function* at  $s$ .

### 3.2.1 Thinking about dependency

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

Think back to our promises for more generalized dynamical systems from Section 1.3. We can begin to think of outputs not only as outward expressions, but as positions that one takes within one’s arena—terminology we’ve been using all along. If the arena is your body, then your outputs would be the positions that your body could take. This includes where you go, as well as the direction you’re looking with your head and eyes, whether your lips are pursed or not, etc. In fact, every form of output you provide, from talking to gesturing to moving another object, is performed by changing your position. And your position determines what inputs you’ll notice: if your eyes are closed, your input type is different than if your eyes are open.

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

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

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

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

should capture every way an external force can independently decide to interact with the system.

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

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

### 3.2.2 Examples of dependent dynamical systems

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

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

$$\varphi: Sy^S \rightarrow y^A + 1.$$

To give such a map, we provide a return function  $\varphi_1: S \rightarrow 2$  (here we are thinking of 2 as the position-set of  $y^A + 1$ ). A function  $\varphi_1$  sends some elements of  $S$  to 1 and others to 2; those that are sent to 2 are said to be accept states.

If we reach an accept state, we want the machine to halt. So at the position 2, corresponding to the summand 1, there are no directions—and thus no inputs available. In other words, the update function  $\varphi_s^\#$  is trivial when  $\varphi_1(s) = 2$ .

On the other hand,  $\varphi_s^\#$  is not trivial on those elements  $s \in S$  for which  $\varphi_1(s) = 1$ . Instead, these update functions  $\varphi_s^\#: A \rightarrow S$  specify how the machine updates its state on each input from  $A$ , if the current state is  $s$ . This is in line with the way the automaton's update function behaves.

When equipped with an initial state  $s_0 \in S$  specified by a map  $y \rightarrow Sy^S$ , we call these dependent dynamical systems *halting deterministic state automata*. Given a sequence (or “word”)  $(a_1, \dots, a_n)$  in  $\text{List}(A)$ , we say that the automaton *accepts* this word if starting at the initial state and following the elements in the sequence leads us to an accept state, *without hitting an accept state and stopping too early*—or, more formally, if there exists a sequence  $(s_0, s_1, \dots, s_n)$  such that  $\varphi_1(s_n) = 2$  and, for all  $k \in [0, n - 1]$ , we have  $\varphi_1(s_k) = 1$  and

$$s_{k+1} = \varphi_{s_k}^\#(a_{k+1}).$$

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

*Remark 3.20.* Again, the conditions for when one of these automata accepts a word are rather awkward to formally state. We will see in Example 7.50 an alternative way of saying whether a word is accepted by a halting deterministic state automaton.

*Exercise 3.21.* Consider the halting deterministic state automaton shown below:



Let the left state  $\bullet$  be 1, the right state  $\bullet$  be 2, and the bottom state  $\bullet$  be 3. We designate  $\bullet$ , state 1, as the initial state. We can also call the orange arrows “orange” and the green arrows “green.” Answer the following questions, in keeping with the notation from Example 3.19.

1. What is  $S$ ?
2. What is  $A$ ?
3. Based on the labeled transition diagram, which states are accept states, and which are not?
4. Specify the corresponding lens  $Sy^S \rightarrow y^A + 1$ .
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?  $\diamond$

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

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

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

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

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

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

The graph itself can be seen as a dynamical system  $\varphi: Vy^V \rightarrow g$ , where  $\varphi_1 = \text{id}_V$  and  $\varphi_v^\#(e) = t(e)$ . So its states are the vertices of the graph, each vertex returns itself as output, and an input at a vertex  $v \in V$  is an edge  $e \in E$  coming out of that vertex that takes us from the vertex  $v = s(e)$  along the edge  $e$  to its target vertex  $\varphi_v^\#(e) = t(e)$ .

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

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

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

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

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

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

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

We want our file-reader to be ‘ready’ if its current entry is the terminal entry; otherwise, it will be ‘busy.’ In either case, it will return the  $\text{ascii}$  character at the current position. So we define the return function  $\varphi_1$  such that, for all  $(s, t) \in S$ ,

$$\varphi_1(s, t) = \begin{cases} (\text{'ready'}, f(s)) & \text{if } s = t \\ (\text{'busy'}, f(s)) & \text{otherwise} \end{cases}$$

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

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

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

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

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

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

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

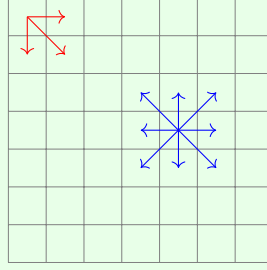
Then we can model a robot that can be instructed to move within an  $n \times n$  grid as a dependent dynamical system  $\varphi: Sy^S \rightarrow p$ , with  $S := n \times n$  and

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

The robot's state is a position in the grid. We let  $\varphi_1 := \text{id}_{n \times n}$  so that each state returns itself as output—in particular, the robot returns a position as output by moving to that position in the grid.

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

the center of the grid than when it is on the sides or corners:



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

*Exercise 3.29.* Modify the dynamical system from Example 3.28 as follows.

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

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

### 3.3 Constructing new dynamical systems from old

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

1. Given dynamical systems  $Sy^S \rightarrow p$  and  $Sy^S \rightarrow q$ , we can use the universal property of the *categorical product* to obtain a dynamical system  $Sy^S \rightarrow p \times q$ ; see Section 3.3.1.



2. Given dynamical systems  $\varphi: Sy^S \rightarrow p$  and  $\psi: Ty^T \rightarrow q$ , we can take their parallel product to obtain a dynamical system  $\varphi \otimes \psi: STy^{ST} \rightarrow p \otimes q$ ; see Section 3.3.2.
3. Given a dynamical system  $\varphi: Sy^S \rightarrow p$  and a lens  $f: p \rightarrow q$ , we can compose them to obtain a dynamical system  $\varphi \circ f: Sy^S \rightarrow q$ ; see Section 3.3.3.

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

### 3.3.1 Categorical products: multiple interfaces operating on the same states

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

$$\varphi: Sy^S \rightarrow \prod_{i \in n} p_i,$$

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

We can characterize the dynamics of  $\varphi$  in terms of each original dynamical system  $\varphi_i$  as follows. By Exercise 2.88, the return function

$$\varphi_1: S \rightarrow \prod_{i \in n} p_i(1)$$

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

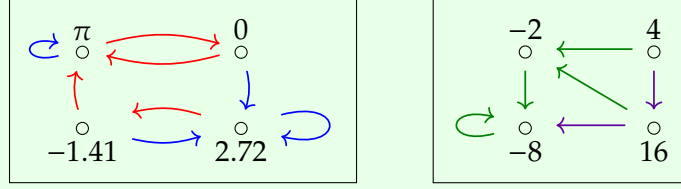
$$\varphi_s^\sharp: \sum_{i \in n} p_i[(\varphi_i)_1(s)] \rightarrow S$$

sends each pair  $(i, d)$  in its domain, with  $i \in n$  and  $d \in p_i[(\varphi_i)_1(s)]$ , to where the update function of  $\varphi_i$  at  $s$  sends  $d$ : namely  $(\varphi_i)_s^\sharp(d)$ .

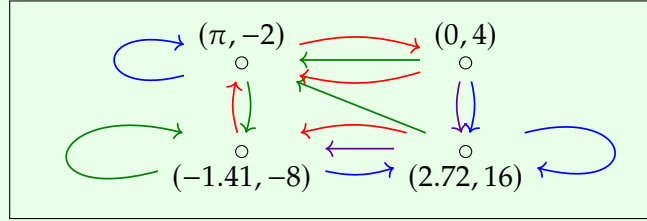
In other words, if there are multiple interfaces that can drive the same set of states, we may view them as a single product interface that can drives those states. This single dynamical system returns output in all of the original systems at once; then it can receive input from any one of the original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

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

*Example 3.31.* Consider two four-state dependent dynamical systems  $\varphi: 4y^4 \rightarrow \mathbb{R}y^{\{r,b\}}$  and  $\psi: 4y^4 \rightarrow \mathbb{Z}_{\geq 0}y^{\{g,p\}} + \mathbb{Z}_{< 0}y^{\{g\}}$ , 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  $4y^4 \rightarrow \mathbb{R}Z_{\geq 0}y^{\{r,b,g,p\}} + \mathbb{R}Z_{< 0}y^{\{r,b,g\}}$  that looks like this:



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

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

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

◇

### 3.3.2 Parallel products: juxtaposing dynamical systems

Another way to combine two polynomials—and indeed two lenses—is by taking their parallel product, as in Section 2.4.2. In particular, the parallel product of two state

systems is still a state system. So parallel products give us another way to create new dynamical systems from old ones. In fact, it's about as easy as you could hope: you just multiply the set of states, multiply the set of outputs, and multiply the set of inputs at each of those outputs.

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

$$\varphi: \left( \prod_{i \in n} S_i \right) y^{\prod_{i \in n} S_i} \cong \bigotimes_{i \in n} S_i y^{S_i} \rightarrow \bigotimes_{i \in n} p_i,$$

which is itself a dynamical system.

We can characterize the dynamics of  $\varphi$  in terms of each constituent dynamical system  $\varphi_i$  as follows. By the proof of Proposition 2.97, the return function

$$\varphi_1: \prod_{i \in n} S_i \rightarrow \prod_{i \in n} p_i(1)$$

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

$$\varphi_{(s_i)_{i \in n}}^\sharp: \prod_{i \in n} p_i[(\varphi_i)_1(s_i)] \rightarrow \prod_{i \in n} S_i$$

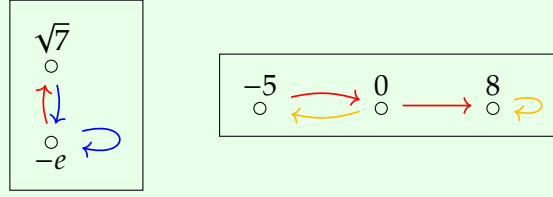
sends each  $n$ -tuple of directions  $(d_i)_{i \in n}$  in its domain, with each  $d_i \in p_i[(\varphi_i)_1(s_i)]$ , to the  $n$ -tuple  $((\varphi_i)_{s_i}^\sharp(d_i))_{i \in n}$  consisting of states where the update function of each  $\varphi_i$  at  $s_i$  sends  $d_i$ .

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

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

*Example 3.34.* Consider two dependent dynamical systems  $\varphi: 2y^2 \rightarrow \mathbb{R}_{<0}y^{\{b,r\}} + \mathbb{R}_{\geq 0}y^{\{b\}}$  and  $\psi: 3y^3 \rightarrow \mathbb{Z}_{<0}y^{\{r\}} + \{0\}y^{\{r,y\}} + \mathbb{Z}_{>0}y^{\{y\}}$ , drawn below as labeled transition

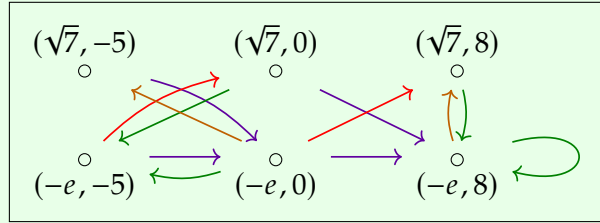
diagrams (we think of  $b$ ,  $r$ , and  $y$  as blue, red, and yellow, respectively):



Taking their parallel product, we obtain a dynamical system with state system  $6y^6$  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—really a pair of states from the constituent state sets—returns two outputs, one according to the return function of  $\varphi$  and another according to the return function of  $\psi$ . Then every input must be a pair of inputs from the constituent state sets, with the update function updating each state in the pair given each input in the pair according to the constituent update functions  $\varphi$  and  $\psi$ .

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

◇

*Exercise 3.36.* 1. Explain how the robot-on-a-grid dynamical system  $\varphi: Sy^S \rightarrow p$  from Example 3.28 can be written as the parallel product of some dynamical system with itself.

2. Use  $k$ -fold parallel products to generalize Example 3.28 to robots on  $k$ -dimensional grids.

◇

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

### 3.3.3 Composing lenses: wrapper interfaces

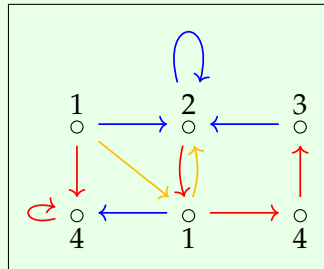
The idea behind a wrapper interface is simple. Given a dynamical system  $\varphi: Sy^S \rightarrow p$ , say that we wanted to interact with it using a new interface  $q$  rather than  $p$ . We can do this as long as we have a lens  $f: p \rightarrow q$ , which we can then compose with our original dynamical 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  $\varphi$  (or sometimes just  $p$ , if a dynamical system  $\varphi$  has yet to be specified) using  $f$ .

To see how this new composite system  $\varphi \circ f$  relates to the original dynamical system  $\varphi$ , it is best to view  $f$  as a delegation of decisions like we did in Section 2.3.2. The lens  $f$  converts an output  $i$  from  $p$  to an output  $f_1(i)$  from  $q$  on positions, but at the same time is allowing the decision of which input in  $p[i]$  to choose next to depend on a choice of input from  $q[f_1(i)]$  instead, via its on-directions function at  $i$ . So the wrapper  $f$  converts output from the original interface  $p$  to output from the wrapper interface  $q$ , and it converts input into the wrapper interface  $q$  to input into the original interface  $p$ . Precomposed with a dynamical system, we obtain a new dynamical system that allows you to interact with the original system using only this new interface wrapped around it.

*Example 3.37.* Consider a dependent dynamical system  $\varphi: 6y^6 \rightarrow p$  with

$$p := \{1\}y^{\{b,y,r\}} + \{2\}y^{\{b,r\}} + \{3\}y^{\{b\}} + \{4\}y^{\{r\}},$$

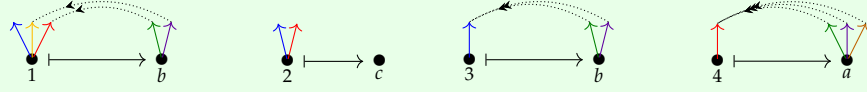
drawn below as a labeled transition diagram (we think of  $b, y$ , and  $r$  as blue, yellow, and red, respectively):



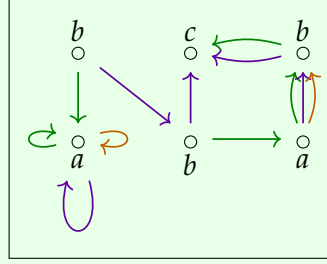
We will wrap the interface

$$q := \{a\}y^{\{g,p,o\}} + \{b\}y^{\{g,p\}} + \{c\}$$

around  $\varphi$  using the following lens  $f: p \rightarrow q$  (we think of  $g, p$ , and  $o$  as green, purple, and orange, respectively):



Composing  $\varphi$  with  $f$ , we obtain a dynamical system  $6y^6 \xrightarrow{\varphi} p \xrightarrow{f} q$  that looks like this:



Each state returns an output in  $q$  according to where the on-positions function of  $f$  sends the output the state returns in  $p$ . Then each input in  $q$  is passed to an input in  $p$  via the corresponding on-directions function of  $f$ , whereupon the update function of  $\varphi$  computes the new state. So  $f$  allows us to operate  $\varphi$  with the wrapper interface  $q$  instead of the original interface  $p$ .

*Exercise 3.38.* In Exercise 3.27, you constructed a file-searcher  $\psi: Sy^S \rightarrow q$  by taking the file-reader  $\varphi: Sy^S \rightarrow p$  from Example 3.26 and replacing its interface  $p$  with a new interface  $q$  while keeping its state system  $Sy^S$  the same. Express this construction as wrapping  $q$  around  $\varphi$  by giving a lens  $f: p \rightarrow q$  for which composing  $\varphi$  with  $f$  yields  $\psi$ .  $\diamond$

In the next section, we describe a special kind of wrapper.

### 3.3.4 Situations as enclosures

Say we wanted to model a dynamical system  $\varphi: Sy^S \rightarrow p$  within a closed system, for which an external agent can perceive no change in output and effect no change in input. We can think of this as wrapping  $y$ , the interface with one output and one input, around  $\varphi$ . To do so, we must specify a wrapper  $\gamma: p \rightarrow y$ . We call such a wrapper an *enclosure* for an interface  $p$ , since it is a way of closing off  $p$  to the outside world.

Let us zoom out from the dynamical systems interpretation of polynomials for the time being to examine the categorical properties of enclosures. Given a polynomial  $p$ , we denote the set of lenses  $p \rightarrow y$  by

$$\Gamma(p) := \mathbf{Poly}(p, y) \quad (3.39)$$

as we did in Proposition 1.18. By (2.55), we have that

$$\Gamma(p) \cong \prod_{i \in p(1)} p[i], \quad (3.40)$$

so in terms of arenas, a map  $\gamma \in \Gamma(p)$  can be thought of as a dependent function that assigns each  $p$ -position  $i$  to a direction  $\gamma(i)$  of  $p$  at  $i$ . So we call an element of  $\Gamma(p)$  a *situation* for  $p$ . The idea is that the situation you're in gives you a direction to go along at any position you may take.

Returning to the language of dynamical systems, a situation for  $p$  corresponds to an enclosure for the interface  $p$ , in that it chooses a fixed input at every output of  $p$ . An enclosure for your interface dictates what you'll see (the input you receive) given anything you might do (the output you provide); there is no need for any further outside interference.

*Remark 3.41.* Although they both refer to lenses into  $y$ , we will favor the term *enclosure* in the context of dynamical systems and wrappers and the term *situation* more generally.

*Exercise 3.42.* Let  $\varphi: Sy^S \rightarrow By^A$  be an  $(A, B)$ -Moore machine.

1. Is it true that an enclosure  $\gamma: By^A \rightarrow y$  can be identified with a function  $A \rightarrow B$ ?
2. Describe how to interpret an enclosure  $\gamma: By^A \rightarrow y$  as a wrapper around an interface  $By^A$ .
3. Given an enclosure  $\gamma$ , describe the dynamics of the composite Moore machine  $Sy^S \xrightarrow{\varphi} By^A \xrightarrow{\gamma} y$  obtained by wrapping  $y$  around  $\varphi$  using  $\gamma$ .  $\diamond$

*Example 3.43* (The do-nothing enclosure). There is something rather off-putting about the way we model dynamical systems as lenses  $\varphi: Sy^S \rightarrow p$ . We know that  $\varphi$  tells us how state-positions return output-positions and, given a current state-position, how input-directions update state-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 state-positions and state-directions for us; we have to bank 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**. Here's where situations save the day: a lens  $Sy^S \rightarrow y$  is just a way of assigning each position in  $S$  to a direction in  $S$ . So we can define  $\epsilon: Sy^S \rightarrow y$  to be the situation that assigns each position  $s \in S$  to the direction  $s$  at  $s$  corresponding to the same state. Note that  $\epsilon$  can be identified with the identity function on  $S$  (see Exercise 3.42). Now **Poly** knows which direction is associated with the same state as the position it is at.

In this way, we can generalize our notion of state systems to monomials  $Sy^{S'}$  equipped with a bijection  $S \rightarrow S'$ , which we can then translate to a situation  $\epsilon: Sy^{S'} \rightarrow y$ . 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 enclosure  $\epsilon: Sy^S \rightarrow y$  acts as a very special (if rather unexciting) dynamical system: it is the *do-nothing enclosure*, with only one possible output and one possible input that always keeps the current state the same. While the system does, well, 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 enclosure on it.<sup>a</sup>

Yet this isn't the whole story. The do-nothing enclosure knows, at each position, the direction that keeps the system at the same state; but it doesn't know which of the other directions send the system to which of the other positions' states. We're 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 with a do-nothing enclosure matching  $1' \mapsto 1, 2' \mapsto 2, 3' \mapsto 3$  to make the first one into a state system, we don't have a way to tell **Poly** how to make the second one a state system yet.

From another perspective,  $\epsilon: Sy^S \rightarrow y$  does nothing, while  $\varphi: Sy^S \rightarrow p$  does “one thing”: it steps through the system once, generating the current state's output with the return function and taking in input with the update function. It's all set 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'll develop the machinery to answer these questions over the course of the three chapters in Part II, starting in Section 5.1.4.

<sup>a</sup>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  $Sy^S \rightarrow p$ . The existence of the do-nothing enclosure explains why our terminology does not clash: every polynomial  $Sy^S$  comes equipped with a dependent dynamical system  $\epsilon: Sy^S \rightarrow y$ , so it really is a state *system*. Later on we'll see other systems that come with  $Sy^S$  for free.

*Exercise 3.44.* The notation  $\Gamma(p)$  for the set of situations for a polynomial  $p$  comes from the common mathematical concept of a “global section.” Show that situations  $\gamma: p \rightarrow y$  are precisely *sections* (as defined in Exercise 2.19) of the canonical function  $\pi_p: \dot{p}(1) \rightarrow p(1)$  from Exercise 2.66.  $\diamond$

**Proposition 3.45.** The situations functor  $\Gamma: \mathbf{Poly} \rightarrow \mathbf{Set}^{\text{op}}$  sends  $(0, +)$  to  $(1, \times)$ :

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

Technically, one could say that  $\Gamma$  preserves coproducts, since coproducts in  $\mathbf{Set}^{\text{op}}$  are products in  $\mathbf{Set}$ .



*Exercise 3.46.* Prove Proposition 3.45. ◇

*Remark 3.47.* The situations functor  $\Gamma: \mathbf{Poly} \rightarrow \mathbf{Set}^{\text{op}}$  is also normal lax monoidal in the sense that there are canonical functions

$$1 \cong \Gamma(y) \quad \text{and} \quad \Gamma(p) \times \Gamma(q) \rightarrow \Gamma(p \otimes q)$$

satisfying certain well-known laws. But we won't need this, so we omit its proof.

## 3.4 General interaction

We now have all the pieces we need to fulfill the promises of Section 1.3 by modeling interactions between dependent dynamical systems, which can change their interfaces and interaction patterns, using **Poly**.

### 3.4.1 Wrapping juxtaposed dynamical systems together

When wrapper interaces are used in conjunction with parallel products, they may encode multiple interacting dynamical systems as a single system. Explicitly, given  $n \in \mathbb{N}$  and  $n$  dynamical systems,  $\varphi_i: S_i y^{S_i} \rightarrow p_i$  for every  $i \in n$ , we can first juxtapose them into a single dynamical system

$$\varphi: \left( \prod_{i \in n} S_i \right) y^{\prod_{i \in n} S_i} \rightarrow \bigotimes_{i \in n} p_i$$

by taking their parallel product. Then we can wrap an interface  $q$  around  $\varphi$  using a wrapper  $f$ , yielding a new dynamical system

$$\left( \prod_{i \in n} S_i \right) y^{\prod_{i \in n} S_i} \xrightarrow{\varphi} \bigotimes_{i \in n} p_i \xrightarrow{f} q.$$

On positions,  $f$  gives a way of combining all the outputs of the constituent interfaces into a single output of the wrapper interace. On directions,  $f$  takes into account the current outputs of each of the constituent interfaces, as well as any new output given to the wrapper interface, then uses all of this to give input to each of the constituent interfaces. In particular, a judiciously chosen on-directions function could feed output from some interfaces as inputs to others. When  $f$  is a wrapper around a parallel product of interfaces, we call  $f$  the *interaction pattern* between those interfaces.

*Example 3.48 (Repeater).* Suppose we have a dependent dynamical system  $\varphi: Sy^S \rightarrow Ay + y$ , which takes unchanging input and sometimes returns elements of  $A$  as output while other times returning only silence. What if we wanted to construct a system  $\psi$  that operates just like  $\varphi$ , but *always* returns elements of  $A$  as output? Where  $\varphi$  would have returned silence, we want  $\psi$  to instead *repeat* the last element of  $A$  that it

returned. (We allow  $\psi$  to repeat an arbitrary element of  $A$  if  $\varphi$  returns silence before it has returned any elements of  $A$  yet.)

Let's think like a programmer. What we need is a way to store an element of  $A$  and then retrieve it. So whenever  $\varphi$  returns an element of  $A$  as output, we store it; then when  $\varphi$  returns silence, we retrieve the last element of  $A$  we stored and return that instead.

What should this storage-retrieval dynamical system look like? It needs to take elements of  $A$  as input, return elements of  $A$  as output, and store elements of  $A$  as states. In fact, the identity lens  $\iota: Ay^A \rightarrow Ay^A$  works perfectly: it returns the element of  $A$  currently stored as output and updates its state to the input it receives.

Now we can juxtapose our original system  $\varphi$  with the storage-retrieval system  $\iota$  by taking their parallel product, yielding a dynamical system

$$\varphi \otimes \iota: SAy^{SA} \rightarrow (Ay + y) \otimes Ay^A$$

that runs both systems simultaneously—and independently. But what we want is for  $\varphi$  and  $\iota$  to interact with each other, and for the resulting system to only output elements of  $A$ . To do so, we need to wrap an interface  $Ay$  around  $\varphi \otimes \iota$  by composing it with some lens

$$f: (Ay + y) \otimes Ay^A \rightarrow Ay,$$

the interaction pattern between the interfaces  $Ay + y$  and  $Ay^A$ , which we must define.

Since  $\otimes$  distributes over  $+$ , it suffices to give maps

$$g: Ay \otimes Ay^A \rightarrow Ay \quad \text{and} \quad h: y \otimes Ay^A \rightarrow Ay.$$

The former corresponds to the case where  $\varphi$  outputs an element of  $A$ , while the latter corresponds to the case where  $\varphi$  is silent.

When  $\varphi$  outputs an element of  $A$ , we want to return that output, but we also want to give that output as input to  $\iota$  so that it can be stored. We don't need to do anything with the output of  $\iota$ ; we can simply discard it. So  $g$  should send  $(a, a') \mapsto a$  on positions, returning the output of  $\varphi$  and discarding the output of  $\iota$ ; and the on-directions function  $g_{(a,a')}^\# : 1 \rightarrow A$  should specify the direction  $a \in A$ , feeding the output of  $\varphi$  as input to  $\iota$ .

Meanwhile, when  $\varphi$  outputs silence, we want to return the output of  $\iota$  instead. We also need to feed the output of  $\iota$  back into  $\iota$  as input so that it can continue to be stored. So  $h$  should be the identity on positions as well as the identity on directions.

*Example 3.49 (Paddling).* Say we wanted to build a Moore machine with interface  $\mathbb{N}y$ ; we may interpret its natural number output as the machine's current location. What if we don't want this machine to jump around wildly? Instead, suppose we want to be very strict about what how far the machine can move and what makes it move.

To accomplish this, we introduce two intermediary systems, which we call the

*paddler* and the *tracker*:<sup>a</sup>

$$\text{paddler}: Sy^S \rightarrow 2y \quad \text{and} \quad \text{tracker}: Ty^T \rightarrow \mathbb{N}y^2$$

The paddler has interface  $2y$  because it is blind (i.e. takes no inputs) and can only move (i.e. output) its paddle to the left side or the right side:  $2 \cong \{\text{left}, \text{right}\}$ . The tracker has interface  $\mathbb{N}y^2$  because it will announce the location of the machine (as an element  $n \in \mathbb{N}$ ) and watch what side the paddler is on (as an element of 2). We can wrap an interface  $\mathbb{N}y$  around them both using an interaction pattern

$$2y \otimes \mathbb{N}y^2 \rightarrow \mathbb{N}y$$

whose on-positions function is the projection  $2\mathbb{N} \rightarrow \mathbb{N}$ , returning the location returned by the tracker, and whose on-directions function is the projection  $2\mathbb{N} \rightarrow 2$ , passing the output of the paddler as input to the tracker.

Let's leave the paddler's dynamics alone—how you make that paddler behave is totally up to you—and instead focus on the dynamics of the tracker. We want it to watch for when the paddle switches from left to right or from right to left; at that moment it should push the paddler forward one unit. Thus the states of the tracker are given by  $T := 2\mathbb{N}$ , storing what side the paddler is on and the current location. The on-positions function of the tracker is the projection  $2\mathbb{N} \rightarrow \mathbb{N}$  that returns the current location; then at each  $(d, i) \in 2\mathbb{N}$ , the on-directions function of the tracker  $2 \rightarrow 2\mathbb{N}$  sends

$$d' \mapsto \begin{cases} (d', i) & \text{if } d = d' \\ (d', i + 1) & \text{if } d \neq d', \end{cases}$$

storing the new direction of the paddler as well as moving the machine forward one unit if the paddle switches while keeping the machine still if the paddle stays still.

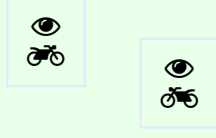
<sup>a</sup>Perhaps one could refer to the tracker as the *demiurge*; it is responsible for maintaining the material universe.

*Exercise 3.50.* Change the dynamics and state system of the tracker in Example 3.49 so that it exhibits the following behavior.

When the paddle switches once and stops, the tracker increases its location by one unit and stops, as before in Example 3.49. But when the paddle switches twice in a row, the tracker increases its location by two units on the second switch! So if it is quiet for a while and then switches three times in a row, the tracker will increase its location by one then two then two.  $\diamond$

*Example 3.51.* Suppose you have two systems with the same interface  $p := q :=$

$\mathbb{R}^2 y^{\mathbb{R}^2 - \{(0,0)\}}$ .



The output of each interface indicates the location of the system, while the range of possible inputs indicate the locations that the system could observe, relative to the location of the system itself. Taking all pairs of reals except  $(0,0)$  corresponds to the fact that the eye cannot see that which is at the same position as the eye.

Let's have the two systems constantly approaching each other with a force equal to the reciprocal of the squared distance between them. If they finally collide, let's have the whole thing come to a halt. To do this, we want the wrapper interface to be  $\{\text{'go'}\}y + \{\text{'stop'}\}$ , so that if the system returns 'go' it can still advance to the next state, but if it returns 'stop' it halts. The wrapper  $\mathbb{R}^2 y^{\mathbb{R}^2 - \{(0,0)\}} \otimes \mathbb{R}^2 y^{\mathbb{R}^2 - \{(0,0)\}} \rightarrow \{\text{'go'}\}y + \{\text{'stop'}\}$  is given on positions by

$$((x_1, y_1), (x_2, y_2)) \mapsto \begin{cases} \text{'stop'} & \text{if } x_1 = x_2 \text{ and } y_1 = y_2 \\ \text{'go'} & \text{otherwise.} \end{cases}$$

On directions, we use the function

$$((x_1, y_1), (x_2, y_2)) \mapsto ((x_2 - x_1, y_2 - y_1), (x_1 - x_2, y_1 - y_2)),$$

so that each system is able to see the location of the other system relative to its own, i.e. the vector pointing from itself to the other system (unless that vector is zero, in which case the whole thing should have already halted).

We can use these vectors to define the internal dynamics of each system so that they move the way we want them to. Each system will hold as its internal state its current location and velocity, i.e.  $S = \mathbb{R}^2 \times \mathbb{R}^2$ . To define a lens  $S y^S \rightarrow \mathbb{R}^2 y^{\mathbb{R}^2 - \{(0,0)\}}$  we simply return the current location, update the current location by adding the current velocity, and update the current velocity by adding a vector with appropriate magnitude pointing to the other system:

$$\begin{aligned} \mathbb{R}^2 \times \mathbb{R}^2 &\xrightarrow{\text{return}} \mathbb{R}^2 \\ ((x, y), (v_x, v_y)) &\xrightarrow{\text{return}} (x, y) \\ \mathbb{R}^2 \times \mathbb{R}^2 \times (\mathbb{R}^2 - \{(0,0)\}) &\xrightarrow{\text{update}} \mathbb{R}^2 \times \mathbb{R}^2 \\ ((x, y), (v_x, v_y), (a, b)) &\xrightarrow{\text{update}} \left( x + v_x, y + v_y, v_x + \frac{a}{(a^2 + b^2)^{3/2}}, v_y + \frac{b}{(a^2 + b^2)^{3/2}} \right) \end{aligned}$$

*Exercise 3.52.* Suppose  $(X, d)$  is a metric space, i.e.  $X$  is a set of points and  $d: X \times X \rightarrow \mathbb{R}_{\geq 0}$  is a distance function satisfying the usual laws. Let's have robots interact in this space.

Let  $A, A'$  be sets, each thought of as a set of signals, and let  $a_0 \in A$  and  $a'_0 \in A'$  be elements, each thought of as a default value. Let  $p := AXy^{A'X}$  and  $p' := A'Xy^{AX}$ , and imagine there are two robots, one with interface  $p$ , returning a signal as an element of  $A$  and its location as a point in  $X$ , and one with interface  $p'$ , returning a signal as an element of  $A'$  and also its location as a point in  $X$ .

1. Write down an interaction pattern  $p \otimes p' \rightarrow y$  such that each robot receives the other's location, but that it only receives the other's signal when their locations  $x, x'$  are sufficiently close, namely when  $d(x, x') < 1$ . Otherwise, it receives the default signal.
2. Write down an interaction pattern  $p \otimes p' \rightarrow y^{[0,5]}$  where the value  $s \in [0, 5]$  is a scalar, allowing the signal to travel  $s$  times further.
3. Suppose that each robot has a set  $S, S'$  of possible private states in addition to their locations. What functions are involved in providing a dynamical system  $\varphi: SXy^{SX} \rightarrow AXy^{A'X}$ , 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.)  $\diamond$

### 3.4.2 Enclosing juxtaposed dynamical systems together

We saw in Section 3.3.4 that a situation (i.e. lens into  $y$ ) for the interface of a dynamical system encloses that dynamical system in a closed system. So it should not come as a surprise that a situation for a parallel product of interfaces yields an interaction pattern between the interfaces that only allows the interfaces to interact with each other, cutting off any other interaction with the outside world.

*Example 3.53* (Picking up the chalk). Imagine that you see some chalk and you pinch it between your thumb and forefinger. An amazing thing about reality is that you will then have the chalk, in the sense that you can move it around. How might we model this in **Poly**? We will construct a closed dynamical system—one with interface  $y$ —consisting of only you and the chalk. To do so, we will provide an interface for you, and interface for the chalk, and an enclosure for your juxtaposition.

Let's say that your hand can be at one of two heights, down or up, and that you can either press (apply pressure between your thumb and forefinger) or not press. Let's also say that you take in information about the chalk's height. Here are the two sets

we'll be using:

$$H := \{\text{'down'}, \text{'up'}\} \quad \text{and} \quad P := \{\text{'press'}, \text{'no press'}\}.$$

Your interface is  $HPy^H$ : returning your own height and pressure, and receiving the chalk's height.

As for the chalk, it is either 'in' your possession or 'out' of it. Either way, it also returns its height, which is either 'down' or 'up' in the air. The chalk always takes in information about whether pressure is being applied or not. When it's 'out' of your possession, that's the whole story, but when it is 'in' your possession, it also receives your hand's height. All together, here are the two interfaces:

$$\text{You} := HPy^H \quad \text{and} \quad \text{Chalk} := \{\text{'out'}\}Hy^P + \{\text{'in'}\}Hy^{HP}.$$

Now we want to give the interaction pattern between you and the chalk. As we said before, you see the chalk's height. If your hand is not at the height of the chalk, the chalk receives no pressure. Otherwise, your hand is at the height of the chalk, so the chalk receives your pressure (or lack thereof). Furthermore, if the chalk is in your possession, it also receives your hand's height.

To provide a map  $\gamma: \text{You} \otimes \text{Chalk} \rightarrow y$ , we use the fact that Chalk is a sum and that  $\otimes$  distributes over  $+$ . Thus we need to give two maps

$$\alpha: HPy^H \otimes Hy^P \rightarrow y \quad \text{and} \quad \beta: HPy^H \otimes Hy^{HP} \rightarrow y$$

The map  $\beta$ , corresponding to when the chalk is in your possession, is quite easy to describe; it can be unfolded to a function  $HPH \rightarrow HHP$ , and we take it to be the obvious map sending your height and pressure to the chalk and the chalk's height to you; see Exercise 3.55. But  $\alpha$  is more semantically interesting: it is given by the map

$$(h_{\text{You}}, p_{\text{You}}, h_{\text{Chalk}}) \mapsto \begin{cases} (h_{\text{Chalk}}, \text{'no press'}) & \text{if } h_{\text{You}} \neq h_{\text{Chalk}} \\ (h_{\text{Chalk}}, p_{\text{You}}) & \text{if } h_{\text{You}} = h_{\text{Chalk}}. \end{cases}$$

So now we've got you and the chalk in enclosed together by  $\gamma$ , so we are ready to add some dynamics. Your dynamics can be whatever you want, so let's just add some dynamics to the chalk (you'll get to give yourself some dynamics in Exercise 3.55). The chalk has only four states  $C := \{\text{'out'}, \text{'in'}\} \times H \cong 4$ : the  $H$  coordinate is its current height, and the other coordinate is whether or not it is in your possession. We will give a dynamical system  $Cy^C \rightarrow \text{Chalk}$  with states  $C$  and interface Chalk, i.e. a lens

$$\{\text{'out'}, \text{'in'}\} \times Hy^{\{\text{'out'}, \text{'in'}\} \times H} \rightarrow \{\text{'out'}\}Hy^P + \{\text{'in'}\}Hy^{HP}. \quad (3.54)$$

On positions, as you might guess, the chalk returns its height and whether it is in your possession directly. On directions, if it's not in your possession, it falls down unless

you catch it (i.e. apply pressure to it so that it enters your possession); if it is in your possession, it takes whatever height you give it. So we can express the on-directions function of (3.54) at ('out',  $h_{\text{Chalk}}$ ) as

$$\begin{aligned} \text{'no press'} &\mapsto (\text{'out'}, \text{'down'}) \\ \text{'press'} &\mapsto (\text{'in'}, h_{\text{Chalk}}) \end{aligned}$$

and the on-directions function of (3.54) at ('in',  $h_{\text{Chalk}}$ ) as

$$\begin{aligned} (h_{\text{You}}, \text{'no press'}) &\mapsto (\text{'out'}, h_{\text{You}}) \\ (h_{\text{You}}, \text{'press'}) &\mapsto (\text{'in'}, h_{\text{You}}). \end{aligned}$$

Obviously, this is all quite complicated, intricate, and contrived. Our goal here is to show that you can define interactions in which one system can engage with or disengage from another, where one system controls the behavior of the other when the two are engaged.

*Exercise 3.55.* 1. In Example 3.53, we said that  $\beta: HPy^H \otimes Hy^{HP} \rightarrow y$  was easy to describe and given by a function  $HPH \rightarrow HHP$ . Explain what's being said, and provide the function.  
2. Provide dynamics to the You interface (i.e. specify a dynamical system with interface You) so that you repeatedly reach down and grab the chalk, lift it with your hand, and drop it.  $\diamond$

Given  $n \in \mathbb{N}$  and polynomials  $p_1, \dots, p_n$  as interfaces, a situation  $p_1 \otimes \dots \otimes p_n \rightarrow y$  puts these  $n$  interfaces in an enclosure together. The following proposition provides an alternative perspective on such situations.

**Proposition 3.56.** Given polynomials  $p, q \in \mathbf{Poly}$ , there is a bijection

$$\Gamma(p \otimes q) \cong \mathbf{Set}(q(1), \Gamma(p)) \times \mathbf{Set}(p(1), \Gamma(q)). \quad (3.57)$$

The idea is that specifying an enclosure for interfaces  $p$  and  $q$  together is equivalent to specifying an enclosure for  $p$  for every output  $q$  might return and specifying an enclosure for  $q$  for every output  $p$  might return.

*Proof of Proposition 3.56.* This is a direct calculation:

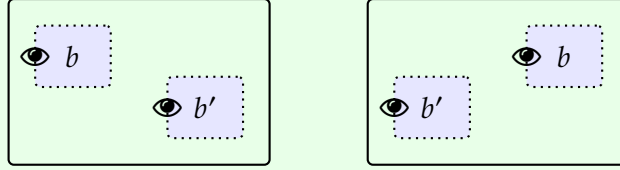
$$\begin{aligned} \Gamma(p \otimes q) &\cong \prod_{i \in p(1)} \prod_{j \in q(1)} (p[i] \times q[j]) \\ &\cong \left( \prod_{j \in q(1)} \prod_{i \in p(1)} p[i] \right) \times \left( \prod_{i \in p(1)} \prod_{j \in q(1)} q[j] \right) \end{aligned}$$

$$\cong \mathbf{Set}(q(1), \Gamma(p)) \times \mathbf{Set}(p(1), \Gamma(q)).$$

□

*Example 3.58.* An enclosure  $f: By^A \otimes B'y^{A'} \rightarrow y$ , corresponds to a map  $BB' \rightarrow AA'$ . In other words, for every pair of outputs  $(b, b') \in BB'$ , the enclosure  $f$  specifies a pair of inputs  $(a, a') \in AA'$ .

Let's think of elements of  $B$  and  $B'$  not as outputs, but as locations that two machines may occupy.



Then given a pair of locations  $(b, b')$ , the interaction pattern  $f$  tells us what the two eyes see, i.e. what values of  $(a, a')$  they get. Equivalently, (3.57) says that the interaction pattern tells us what values the first eye sees at any location when the second eye's location is fixed at  $b'$ , as well as what values the second eye sees at any location when the first eye's location is fixed at  $b$ .

Here we see that (3.57) provides two ways to interpret the interaction pattern between two interfaces in a closed system: either as an enclosure around each interface that the other is part of, or as a single enclosure around them both.

*Exercise 3.59.* Let  $p := q := \mathbb{N}y^{\mathbb{N}}$ . We wish to specify an enclosure around their juxtaposition.

1. Say we wanted to feed the output of  $q$  as input to  $p$ . What function  $f: q(1) \rightarrow \Gamma(p)$  captures this behavior?
2. Say we wanted to feed the sum of the outputs of  $p$  and  $q$  as input to  $q$ . What function  $g: p(1) \rightarrow \Gamma(q)$  captures this behavior?
3. What enclosure  $\gamma: p \otimes q \rightarrow y$  does the pair of functions  $(f, g)$  correspond to via (3.57)?
4. Let dynamical systems  $\varphi: \mathbb{N}y^{\mathbb{N}} \rightarrow p$  and  $\psi: \mathbb{N}y^{\mathbb{N}} \rightarrow q$  both be the identity on  $\mathbb{N}y^{\mathbb{N}}$ . Suppose  $\varphi$  starts in the state  $0 \in \mathbb{N}$  and  $\psi$  starts in the state  $1 \in \mathbb{N}$ . Describe the behavior of the system obtained by enclosing  $\varphi$  and  $\psi$  together with  $\gamma$ , i.e. the system  $(\varphi \otimes \psi) \circ \gamma$ . ◇

*Exercise 3.60.* We will use (3.57) to consider the interaction pattern  $\gamma$  between You and Chalk from Example 3.53 as a pair of functions  $\text{You}(1) \rightarrow \Gamma(\text{Chalk})$  and  $\text{Chalk}(1) \rightarrow \Gamma(\text{You})$ .

1. How does the chalk's output specify an enclosure for you? That is, write the map



$\text{Chalk}(1) \rightarrow \Gamma(\text{You})$ .

2. How does your output specify an enclosure for the chalk? That is, write the map  $\text{You}(1) \rightarrow \Gamma(\text{Chalk})$ .  $\diamond$

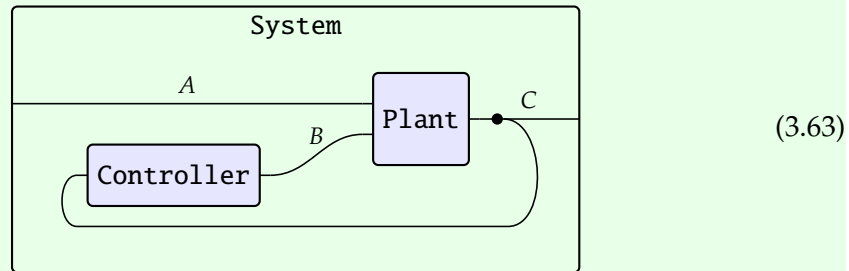
- Exercise 3.61.* 1. State and prove a generalization of (3.57) from Proposition 3.56 for  $n$ -many polynomials  $p_1, \dots, p_n \in \mathbf{Poly}$ .  
 2. Generalize the “idea” statement between Proposition 3.56 and its proof.  $\diamond$

### 3.4.3 Wiring diagrams as interaction patterns

We first saw interactions between systems drawn as wiring diagrams in Section 1.3. They depict systems as boxes, showing how they can send inputs and outputs to each other through the wires between them, as well as how multiple systems can combine to form a larger system whenever smaller boxes are nested within a larger box.

Formally, and more precisely, we can think of each box in a wiring diagram as an interface given by some monomial. The box itself is not, per se, a dynamical system as we have defined one; but it becomes a dynamical system once we equip it with a lens from a state system. Then the entire wiring diagram—specifying how these boxes nest within a larger box—is just an interaction pattern between the interfaces, with the larger box playing the role of the wrapper interface. Once every nested box is equipped with a lens from a state system, we obtain a dynamical system whose interface is the larger box.

*Example 3.62.* Here is a simple wiring diagram.



The plant is receiving information from the world outside the system, as well as from the controller. It’s also producing information for the outside world which is being monitored by the controller.

There are three boxes shown in (3.63): the controller, the plant, and the system. Each has a fixed set of inputs and outputs, and so we can consider the box as a monomial interface.

$$\text{Plant} := Cy^{AB} \quad \text{Controller} := By^C \quad \text{System} := Cy^A. \quad (3.64)$$

The wiring diagram itself is a wrapper

$$w: \text{Plant} \otimes \text{Controller} \rightarrow \text{System},$$

specifying an interaction pattern between **Plant** and **Controller** with wrapper interface **System**. Concretely,  $w$  is a lens  $CB y^{ABC} \rightarrow Cy^A$  that dictates how wires are feeding from outputs to inputs. Like all lenses between monomials,  $w$  consists of an on-positions function  $CB \rightarrow C$  and an on-directions function  $CBA \rightarrow ABC$ .

The wiring diagram is a picture that tells us which maps to use. The on-positions function says “inside the system you have boxes outputting values of type  $C$  and  $B$ . The system needs to produce an output of type  $C$ ; how shall I obtain it?” The answer, according to the wiring diagram, is to send  $(c, b) \mapsto c$ .

Meanwhile, the on-directions function says “inside the system you have boxes outputting values of type  $C$  and  $B$ , and the system itself is receiving an input value of type  $A$ . The boxes inside need input values of type  $A$ ,  $B$ , and  $C$ ; how shall I obtain them?” Again, we can read the answer off the wiring diagram: send  $(c, b, a) \mapsto (a, b, c)$ .

Note that neither the wiring diagram nor any of the boxes within it are dynamical systems on their own. Rather, each box is a monomial that could be the interface of a dynamical system. When we assign to a box a dynamical system having that box as its interface, we say that *give dynamics* to the box. So the entire wiring diagram is a wrapper that tells us how, given the dynamics for each inner box,

$$\varphi: Sy^S \rightarrow \text{Plant} \quad \text{and} \quad \psi: Ty^T \rightarrow \text{Controller},$$

we can obtain the dynamics for the outer box:

$$STy^{ST} \xrightarrow{\varphi \otimes \psi} \text{Plant} \otimes \text{Controller} \xrightarrow{w} \text{System}.$$

- Exercise 3.65.* 1. Make a new wiring diagram like (3.63) except where the controller also receives information of type  $A'$  from the outside world.
2. What are the monomials represented by the boxes in your diagram (replacing (3.64))?
3. What is the interaction pattern represented by this wiring diagram? Give the corresponding lens, including its on-positions and on-directions functions.  $\diamond$

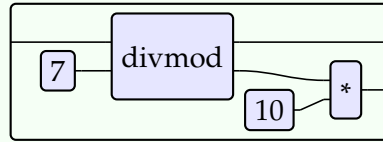
*Exercise 3.66.* Consider the following wiring diagram.



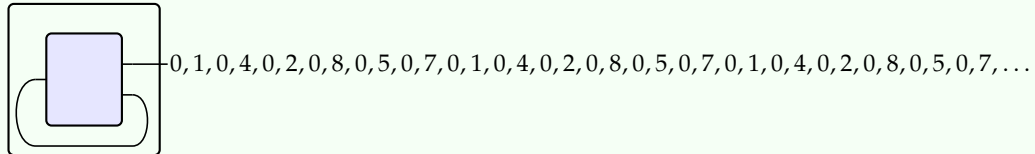
1. Write out the polynomials for each of Alice, Bob, and Carl.
2. Write out the polynomial for the outer box, Team.
3. The wiring diagram constitutes a lens  $f$  in **Poly**; what is its domain and codomain?
4. What lens is it?
5. Suppose we have dynamical systems  $\alpha: Ay^A \rightarrow \text{Alice}$ ,  $\beta: By^B \rightarrow \text{Bob}$ , and  $\gamma: Cy^C \rightarrow \text{Carl}$ . What is the induced dynamical system with interface Team?  $\diamond$

*Exercise 3.67 (Long division).* 1. Let  $\text{divmod}: \mathbb{N} \times \mathbb{N}_{\geq 1} \rightarrow \mathbb{N} \times \mathbb{N}$  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 Exercise 3.9 to turn it into a dynamical system.

2. Interpret the following wiring diagram, where we have already given dynamics to each box as indicated by their labels:



3. Use the above and a diagram of the following form to create a dynamical system that alternates between spitting out 0's and the base-10 digits of  $1/7$  after the decimal point, like so:



We will see in Section 6.1.5 how to make a dynamical system run twice as fast, then apply this to the above system in Example 6.13 so that it skips the 0's.  $\diamond$

*Example 3.68 (Graphs as wiring diagrams and cellular automata).* Suppose we have a graph  $G = (E \rightrightarrows V)$  as in Definition 3.23 and a set  $\tau(v)$  associated with each vertex

$v \in V$ :

$$E \begin{array}{c} \xrightarrow{s} \\ \xleftarrow{t} \end{array} V \xrightarrow{\tau} \mathbf{Set}$$

We can think of  $G$  as an alternative representation of a specific kind of wiring diagram, one where each inner box has exactly one output wire and the outer box is closed. The vertices  $v \in V$  are the inner boxes, the set  $\tau(v)$  is the set associated with  $v$ 's output wire, and each edge  $e$  is a wire connecting the output wire of its target  $t(e)$  to an input wire of its source  $s(e)$ . An edge from a vertex  $v_0$  to a vertex  $v_1$  indicates that the inputs to  $v_0$  depend on the outputs of  $v_1$ .<sup>a</sup>

In other words, we can associate each vertex  $v \in V$  with the monomial

$$p_v := \tau(v)y^{\prod_{e \in E_v} \tau(t(e))}$$

specifying its inputs and outputs, where  $E_v := s^{-1}(v) \subseteq E$  denotes the set of edges emanating from  $v$ . The graph then determines an enclosure

$$\gamma: \bigotimes_{v \in V} p_v \rightarrow y$$

given by a function

$$\prod_{v \in V} \tau(v) \longrightarrow \prod_{e \in E} \tau(t(e))$$

that sends each dependent function  $o: (v \in V) \rightarrow \tau(v)$  to the dependent function  $(e \in E) \rightarrow \tau(t(e))$  sending  $e \mapsto o(t(e))$ . In other words, given the output  $o(v) \in \tau(v)$  for every vertex  $v \in V$ , we know for each edge  $e \in E$  that the input  $s(e)$  receives is the output of  $t(e)$ .

Hence, once we give dynamics to each  $p_v$ , namely by specifying a dynamical system  $S_v y^{S_v} \rightarrow p_v$  with outputs in  $\tau(v)$  and inputs in  $\prod_{e \in E_v} \tau(t(e))$ , we will obtain a closed dynamical system that transitions from each vertex's state to the next according to the information that they pass each other along their edges.

Effectively, by interpreting a graph as a wiring diagram and giving each vertex dynamics, we have created what is known as a *cellular automaton*—a network of vertices (or *cells*) with states, such that each vertex  $v \in V$  “listens” to the signals its *neighbors* in  $E_v$  send based on their states, then responds accordingly by updating its own state.

For example, a common graph found in cellular automata is a 2-dimensional integer lattice, with vertices  $V := \mathbb{Z} \times \mathbb{Z}$ . The edges indicate which vertices are neighbors and thus “hear” which other vertices. One might use

$$E := (\{-1, 0, 1\} \times \{-1, 0, 1\} - \{(0, 0)\}) \times V$$

with  $s(i, j, m, n) = (m, n)$  and  $t(i, j, m, n) = (m + i, n + j)$ , so that the neighbors of each vertex are the eight vertices that surround it.

<sup>a</sup>We could have defined the edges in the opposite directions, so that they would point in the direction of data flow rather than in the direction of data dependencies; this was an arbitrary choice.

*Exercise 3.69* (Conway’s Game of Life). Conway’s Game of Life is played on a 2-dimensional integer lattice as follows. Each lattice point is either *live* or *dead*, and each point observes its eight *neighbors* to which it is horizontally, vertically, or diagonally adjacent. The following occurs at every time step:

- Any live point with 2 or 3 live neighbors remains live.
- Any dead point with 3 live neighbors becomes live.
- All other points either become or remain dead.

We can use Example 3.68 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  $\tau: V \rightarrow \mathbf{Set}$ ?
3. What are the polynomials  $p_v$  from Example 3.68?
4. What is the appropriate state set  $S_v$  for each interface  $p_v$ ?
5. What is the appropriate dynamical system map  $S_v y^{S_v} \rightarrow p_v$ ?

◇

### 3.4.4 More examples of general interaction

While wiring diagrams are a handy visualization tool for certain simple interaction patterns, there are more general interaction patterns that cannot be captured by such a diagram. For example, here we generalize our previous cellular automata example.

*Example 3.70* (Generalized cellular automata: voting on who your neighbors are). Recall from Example 3.68 how we constructed a cellular automaton on a graph  $G = (E \rightrightarrows V)$ . For each  $v \in V$ , the graph specifies the set  $N(v) := t(E_v)$  of vertices at the ends of edges coming out of  $v$ . These vertices are the *neighbors* of  $v$ , or the vertices that  $v$  can “listen” to. We call the map  $N: V \rightarrow 2^V$  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  $S_v := \tau(v) := 2$ .

Now just take the vertices and forget the edges of our graph. Suppose instead that we are given a function  $n: V \rightarrow \mathbb{N}$  that we think of as specifying the number  $n(v)$  of neighbors each  $v \in V$  could potentially have. Let  $\mathfrak{n}(v) := \{1, 2, \dots, n(v)\}$ . Then we can think of the monomial that each vertex represents as

$$p_v \cong 2y^{2^{n(v)}},$$

returning its own state as output and receiving its potential neighbors’ states as input.

Say that a neighbor function  $N: V \rightarrow 2^V$  *respects*  $n$  if we have an isomorphism  $N(v) \cong \mathfrak{n}(v)$  for each  $v \in V$ . Now suppose we have a function  $N'_\cdot: 2^V \rightarrow (2^V)^V$  that sends each set of vertices  $S \in 2^V$  to a neighbor function  $N'_S: V \rightarrow 2^V$  that respects  $n$ . In other words, each possible state configuration  $S$  of all the vertices in  $V$  determines a neighbor function  $N'_S$ . In the case of Example 3.68, when we had a graph, it told us what

the neighbor function should always be. Now we can think of it like all the vertices are voting, via  $N'$ , on what neighbor function to use to determine which vertices are listening to which others.

We can put this all together by providing an enclosure for all the vertices,

$$\bigotimes_{v \in V} p_v \cong 2^V y^{2^{\sum_{v \in V} n(v)}} \longrightarrow y. \quad (3.71)$$

Specifying such an enclosure amounts to specifying a function  $g: 2^V \rightarrow 2^{\sum_{v \in V} n(v)}$  that assigns each possible state configuration  $S \in 2^V$  of all the vertices in  $V$  to a function  $g(S): \sum_{v \in V} n(v) \rightarrow 2$  specifying the states every vertex hears. But we already have a neighbor function assigned to  $S$  that respects  $n$ , namely  $N'_S$ , for which  $N'_S(v) \cong n(v)$  for all  $v \in V$ . So we can think of  $g(S)$  equivalently as a function  $g(S): \sum_{v \in V} N'_S(v) \rightarrow 2$  that says for each  $v \in V$  what signal in 2 it should receive from its neighbor  $w \in N'_S(v)$ . But we can just have it receive the current state of its neighbor, as given by  $S$ :

$$g(S)(v, w) := S(w).$$

We have accomplished our goal: the vertices “vote” on how they ought to be connected, in that their states together determine the neighbor function. Of course, we don’t mean to imply that this vote needs to be democratic or fair in any way: it is an arbitrary function  $N'_: 2^V \rightarrow (2^V)^V$ . It could be dictated by a given vertex  $v_0 \in V$  in the sense that its state completely determines the neighbor function  $V \rightarrow 2^V$ ; this would be expressed by saying that  $N'_$  factors as  $2^V \rightarrow 2^{\{v_0\}} \cong 2 \xrightarrow{I_0} (2^V)^V$  for some  $I_0$ .

*Exercise 3.72.* We can change Example 3.70 slightly by replacing the wrapper interface  $y$  with some other interface.

1. First change it to  $Ay$  for some set  $A$  of your choice, and update (3.71) so that the system outputs some aspect of the current state configuration of all the vertices  $S \in 2^V$ .
2. What would it mean to change (3.71) to a map  $\bigotimes_{v \in V} p_v \rightarrow y^A$  for some  $A$ ?  $\diamond$

Finally, we are ready to formalize the examples we previewed in Section 1.3.

*Example 3.73.* Recall the first picture from Example 1.11. We said that when too much force is applied to a material, bonds can break. Let’s simplify the picture a bit.



We will imagine the dependent dynamical systems  $\varphi_1: Sy^S \rightarrow p_1$  and  $\varphi_2: Sy^S \rightarrow p_2$  as

initially connected in space. They experience forces from the outside world, and—for as long as they are connected—they experience forces from each other. More precisely, each interface is defined by

$$p_1 := p_2 := Fy^{FF} + \{\text{'snapped'}\}y^F.$$

Elements of  $F$  will be called *forces*. We need to be able to add and compare forces, i.e. we need  $F$  to be an ordered monoid; let's say  $F = \mathbb{N}$  for simplicity. The idea is that the interface has two kinds of output it can return: either a force  $f_i \in F$  on the other system, at which point it can receive an input in  $FF$  indicating a force acting on the system from its left and another force acting on it from its right; or 'snapped,' indicating that the system is no longer connected to the other system, at which point it only receives a single force in  $F$  from the outside.

The wrapper interface is defined to be

$$p := y^{FF};$$

it takes as input two forces  $(f_L, f_R)$  and returns unchanging output.

Though the systems  $\varphi_1$  and  $\varphi_2$  may be initially connected, if the forces on either one surpass a threshold, that system stops sending and receiving forces from the other. The connection is broken and neither system ever receives forces from the other again. This is what we will implement explicitly below.

To do so, we need to define an interaction pattern  $p_1 \otimes p_2 \rightarrow p$  that wraps  $p$  around  $\varphi_1$  and  $\varphi_2$ . That is, we need to give a lens

$$\kappa: (Fy^{FF} + \{\text{'snapped'}\}y^F) \otimes (Fy^{FF} + \{\text{'snapped'}\}y^F) \rightarrow y^{FF}.$$

By the distributivity of  $\otimes$  over  $+$ , it suffices to give four maps:

$$\begin{aligned} \kappa_{11}: FFy^{(FF)(FF)} &\rightarrow y^{FF} \\ \kappa_{12}: F\{\text{'snapped'}\}y^{(FF)F} &\rightarrow y^{FF} \\ \kappa_{21}: \{\text{'snapped'}\}Fy^{F(FF)} &\rightarrow y^{FF} \\ \kappa_{22}: \{\text{'snapped'}\}\{\text{'snapped'}\}y^{FF} &\rightarrow y^{FF} \end{aligned} \tag{3.74}$$

The middle two maps  $\kappa_{12}$  and  $\kappa_{21}$  won't actually occur in our dynamics, so we take them to be arbitrary. We take the last map  $\kappa_{22}$  to be the obvious isomorphism, passing the forces from outside to the two internal interfaces. The first map  $\kappa_{11}$  is equivalent to a function  $(FF)(FF) \rightarrow (FF)(FF)$  which we take to be  $((f_1, f_2), (f_L, f_R)) \mapsto ((f_L, f_2), (f_1, f_R))$ . While the multiple  $F$ 's may be a little hard to keep track of, what this map says is that if  $\varphi_1$  returns the force  $f_1$  on  $\varphi_2$  as output and  $\varphi_2$  returns the force  $f_2$  on  $\varphi_1$  as output, then  $\varphi_1$  receives the force  $f_2$  from the right as input and  $\varphi_2$  receives the force  $f_1$  from the left as input; and in the meantime the left external force  $f_L$  is given to  $\varphi_1$  on the left, while the right external force is given to  $\varphi_2$  on the right.

Now that we have the interfaces wrapped together, it remains to specify each dynamical system. The states in the two cases will be identical, namely  $S := F + \{\text{'snapped'}\}$ , meaning that at any point the system will either be in the state of applying a force to the other system or not. The dynamical systems themselves will be identical as well, up to a symmetry swapping left and right; let's just define the left system for now. It is given by a lens

$$\varphi_1: (F + \{\text{'snapped'}\})y^{F+\{\text{'snapped'}\}} \rightarrow Fy^{FF} + \{\text{'snapped'}\}y^F$$

which we write as the sum of two maps

$$Fy^{F+\{\text{'snapped'}\}} \rightarrow Fy^{FF} \quad \text{and} \quad \{\text{'snapped'}\}y^{F+\{\text{'snapped'}\}} \rightarrow \{\text{'snapped'}\}y^F.$$

Both maps are the identity on positions, directly returning their current state. The second map corresponds to when the connection is broken, after which the connection should remain broken, so its on-directions function is constant, sending any input to 'snapped.' Meanwhile, the first map corresponds to the case where the systems are still connected; this system receives two input forces and must update its state—the force it applies—accordingly. We let the on-directions function  $F(FF) \rightarrow F + \{\text{'snapped'}\}$  send

$$(f_1, (f_L, f_2)) \mapsto \begin{cases} f_L & \text{if } f_1 + f_2 < 100 \\ \text{'snapped'} & \text{otherwise} \end{cases}$$

Thus, when the sum of forces is high enough, the internal state is updated to the 'snapped' state; otherwise, it is sent to the force it receives from outside, which it is now ready to transfer to the other system.

*Example 3.75.* Recall the second picture from Example 1.11. We want to consider the case of a company that may change its supplier based on its internal state. The company returns two possible outputs, corresponding to who it wants to receive widgets  $W$  from:



So the company has interface  $2y^W$ , and each supplier has interface  $Wy$ . Then an enclosure for the company and the suppliers is just a lens  $2y^W \otimes Wy \otimes Wy \rightarrow y$ , corresponding to a function  $2W^2 \rightarrow W$  given by evaluation. In other words, the company's output determines its supplier.



*Example 3.76.* Recall the third picture from Example 1.11. When someone assembles a machine, their own outputs dictate the interaction pattern of the machine's components.



In order for the above picture to make sense, the output set of unit A should be the same as the input set of unit B. Call this set  $X$ , so that unit A has interface  $Xy$  and unit B has interface  $y^X$ . We fix a default value  $x_0 \in X$  for the input to unit B when it is not connected to unit A. Meanwhile, the person takes no input and dictates whether the units are attached or not, so we give it interface  $2y$ .

Then an enclosure for the person and the units is a lens  $2y \otimes Xy \otimes y^X \rightarrow y$ . The lens  $2Xy^X \rightarrow y$ , corresponding to a function  $2X \rightarrow X$  that maps  $(1, x) \mapsto x_0$  and  $(2, x) \mapsto x$ .

We can easily generalize Example 3.76. 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  $q_1, \dots, q_k$  with wrapper interface  $r$ , and that wrapping it around  $p$  to it is just a larger interaction pattern:

$$\mathbf{Poly}(p, [q_1 \otimes \cdots \otimes q_k, r]) \cong \mathbf{Poly}(p \otimes q_1 \otimes \cdots \otimes q_k, r).$$

In other words, if  $p$  is deciding the interaction pattern between  $q_1, \dots, q_k$  with wrapper interface  $r$ , and gets feedback from that interaction pattern itself, then this is equivalent to an interaction pattern with wrapper interface  $r$  that  $p$  is part of alongside  $q_1, \dots, q_k$  inside of  $r$ .

What it also means is that if you want, you can put a little dynamical system inside of  $[q_1 \otimes \cdots \otimes q_k, r]$  and have it be constantly choosing interaction patterns. Let's see how it works.

### 3.5 Closure of $\otimes$

The parallel monoidal product is closed—we have a monoidal closed structure on  $\mathbf{Poly}$ —meaning that there is a closure operation, which we denote  $[-, -]: \mathbf{Poly}^{\text{op}} \times \mathbf{Poly} \rightarrow \mathbf{Poly}$ , such that there is an isomorphism

$$\mathbf{Poly}(p \otimes q, r) \cong \mathbf{Poly}(p, [q, r]) \quad (3.77)$$

natural in  $p, q, r$ . The closure operation is defined on  $q, r$  as follows:

$$[q, r] := \prod_{j \in q(1)} r \circ (q[j]y) \quad (3.78)$$

Here  $\circ$  denotes standard functor composition; informally,  $r \circ (q[j]y)$  is the polynomial you get when you replace each appearance of  $y$  in  $r$  by  $q[j]y$ . Composition, together with the unit  $y$ , is in fact yet another monoidal structure, as we will see in more depth in Part II.

Before we prove that the isomorphism (3.77) holds naturally, let us investigate the properties of the closure operation, starting with some simple examples.

*Exercise 3.79.* Calculate  $[q, r]$  for  $q, r \in \mathbf{Poly}$  given as follows.

1.  $q := 0$  and  $r$  arbitrary.
2.  $q := 1$  and  $r$  arbitrary.
3.  $q := y$  and  $r$  arbitrary.
4.  $q := A$  for  $A \in \mathbf{Set}$  (constant) and  $r$  arbitrary.
5.  $q := Ay$  for  $A \in \mathbf{Set}$  (linear) and  $r$  arbitrary.
6.  $q := y^2 + 2y$  and  $r := 2y^3 + 3$ .

◇

*Exercise 3.80.* Show that for any polynomials  $p_1, p_2, q$ , we have an isomorphism

$$[p_1 + p_2, q] \cong [p_1, q] \times [p_2, q].$$

◇

*Exercise 3.81.* Show that there is an isomorphism

$$[q, r] \cong \sum_{f: q \rightarrow r} y^{\sum_{j \in q(1)} r[f_1(j)]} \quad (3.82)$$

where the sum is indexed by  $f \in \mathbf{Poly}(q, r)$ .

◇

*Exercise 3.83.* Verify that (2.68) holds.

◇

*Example 3.84.* For any  $A \in \mathbf{Set}$  we have

$$[y^A, y] \cong Ay \quad \text{and} \quad [Ay, y] \cong y^A.$$

More generally, for any polynomial  $p \in \mathbf{Poly}$  we have

$$[p, y] \cong \Gamma(p)y^{p(1)}. \quad (3.85)$$

All these facts follow directly from (3.78).

*Exercise 3.86.* Verify the three facts above.  $\diamond$

*Exercise 3.87.* 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.  $\diamond$

**Proposition 3.88.** With  $[-, -]$  as defined in (3.78), there is a natural isomorphism

$$\mathbf{Poly}(p \otimes q, r) \cong \mathbf{Poly}(p, [q, r]). \quad (3.89)$$

*Proof.* We have the following chain of natural isomorphisms:

$$\begin{aligned} \mathbf{Poly}(p \otimes q, r) &\cong \mathbf{Poly}\left(\sum_{i \in p(1)} \sum_{j \in q(1)} y^{p[i]q[j]}, r\right) \\ &\cong \prod_{i \in p(1)} \prod_{j \in q(1)} \mathbf{Poly}(y^{p[i]q[j]}, r) && \text{(Universal property of coproducts)} \\ &\cong \prod_{i \in p(1)} \prod_{j \in q(1)} r(p[i]q[j]) && \text{(Yoneda lemma)} \\ &\cong \prod_{i \in p(1)} \prod_{j \in q(1)} \mathbf{Poly}(y^{p[i]}, r \circ (q[j]y)) && \text{(Yoneda lemma)} \\ &\cong \mathbf{Poly}\left(\sum_{i \in p(1)} y^{p[i]}, \prod_{j \in q(1)} r \circ (q[j]y)\right) \\ &&& \text{(Universal property of (co)products)} \\ &\cong \mathbf{Poly}(p, [q, r]). \end{aligned}$$

□

*Exercise 3.90.* Show that for any  $p, q$  we have an isomorphism of sets

$$\mathbf{Poly}(p, q) \cong [p, q](1).$$

Hint: you can either use the formula (3.78), or just use (3.89) with the Yoneda lemma and the fact that  $y \otimes p \cong p$ .  $\diamond$

The closure of  $\otimes$  implies that for any  $p, q \in \mathbf{Poly}$ , there is a canonical *evaluation* map

$$\text{eval}: [p, q] \otimes p \longrightarrow q. \quad (3.91)$$

As in any closed monoidal category, such an evaluation map has the universal property that for any  $r \in \mathbf{Poly}$  and map  $f: p \otimes q \rightarrow r$ , there is a unique lens  $f': p \rightarrow [q, r]$  such

that the following diagram commutes:

$$\begin{array}{ccc} p \otimes q & \xrightarrow{f' \otimes q} & [q, r] \otimes q \xrightarrow{\text{eval}} r \\ & \searrow f & \nearrow \end{array}$$

*Exercise 3.92.* Obtain the evaluation map  $\text{eval}: [p, q] \otimes p \longrightarrow q$  from (3.91).

◇

*Exercise 3.93.* 1. For any set  $S$ , obtain the map  $Sy^S \rightarrow y$  whose on-directions map is the identity on  $S$  using  $\text{eval}$  and Example 3.84.

2. Show that four maps in (3.74) from Example 3.73, written equivalently as

$$\begin{aligned} \kappa_{11}: Fy^{FF} \otimes Fy^{FF} &\rightarrow y^F \otimes y^F \\ \kappa_{12}: Fy^{FF} \otimes y^F &\rightarrow y^F \otimes y^F \\ \kappa_{21}: y^F \otimes Fy^{FF} &\rightarrow y^F \otimes y^F \\ \kappa_{22}: y^F \otimes y^F &\rightarrow y^F \otimes y^F, \end{aligned} \tag{3.94}$$

can be obtained by taking the parallel product of identity maps and evaluation maps.

◇

*Example 3.95* (Modeling your environment without knowing what it is). Let's imagine a robot whose interface is an arbitrary polynomial  $p$ . Let's imagine it is part of an interaction pattern

$$f: (q_1 \otimes \cdots \otimes q_n) \otimes p \rightarrow r$$

with some other robots whose interfaces are  $q_1, \dots, q_n$ ; let  $q := (q_1 \otimes \cdots \otimes q_n)$ . The interaction pattern induces a lens  $f': q \rightarrow [p, r]$  such that the original system  $f$  factors through the evaluation  $[p, r] \otimes p \rightarrow r$ .

In other words,  $[p, r]$  holds within it all of the possible ways  $p$  can interact with other systems when they are all wrapped in  $r$ . For example, in the case of  $r := y$ , note that  $[p, y] \cong \prod_{i \in p(1)} p[i]y$ . That is, for each  $p$ -position it produces a direction there, which is just what  $p$  needs as input in a closed system.

Now suppose we were to populate the interface  $p$  with dynamics, a map  $Sy^S \rightarrow p$ . One could aim to choose a set  $S$  along with an interesting map  $g: S \rightarrow \mathbf{Poly}(p, r)$ . Then each state  $s$  would include a guess  $g(s)$  about the state of its environment. This is not the real environment  $q$ , but just the environment as it affects  $p$ , namely  $[p, r]$ . The robot's states model environmental conditions.

*Example 3.96 (Chu &).* Suppose we have polynomials  $p_1, p_2, q_1, q_2, r \in \mathbf{Poly}$  and lenses

$$\varphi_1: p_1 \otimes q_1 \rightarrow r \quad \text{and} \quad \varphi_2: p_2 \otimes q_2 \rightarrow r.$$

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 “ $\varphi_1$  with  $\varphi_2$ ” of the following type:

$$\varphi_1 \& \varphi_2: (p_1 \times p_2) \otimes (q_1 + q_2) \rightarrow r$$

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  $\varphi_1$  or  $\varphi_2$  respectively to get a position in  $r$ ; given a direction there, one returns the corresponding direction in  $q_1$  or  $q_2$  respectively, as well as a direction in  $p_1 \times p_2$  which is either a direction in  $p_1$  or in  $p_2$ .

This sounds complicated, but it can be done formally, once we have monoidal closure. We first rearrange both  $\varphi_1, \varphi_2$  to be  $p$ -centric, using monoidal currying:

$$\psi_1: p_1 \rightarrow [q_1, r] \quad \text{and} \quad \psi_2: p_2 \rightarrow [q_2, r]$$

Now we multiply to get  $\psi_1 \times \psi_2: p_1 \times p_2 \rightarrow [q_1, r] \times [q_2, r]$ . Then we apply Exercise 3.80 to see that  $[q_1, r] \times [q_2, r] \cong [q_1 + q_2, r]$ , and finally monoidal-uncurry to obtain  $(p_1 \times p_2) \otimes (q_1 + q_2) \rightarrow r$  as desired.

### 3.6 Summary and further reading

In this chapter we explained how discrete dynamical systems can be expressed as certain maps of polynomial functors. For example, a Moore machine has an input set  $A$ , an output set  $B$ , a set of states  $S$ , a return function  $S \rightarrow B$ , and an update function  $A \times S \rightarrow S$ . All this is captured in a map of polynomials

$$Sy^S \rightarrow By^A.$$

We discussed a generalization  $Sy^S \rightarrow p$ , where the output is an arbitrary polynomial  $p \in \mathbf{Poly}$ . 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 maps  $p \rightarrow q$ .

Throughout the chapter we gave quite a few different examples. For example, we discussed how every function  $A \rightarrow B$  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>.

---

# 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 just gently leaf through this chapter, to see how well-behaved and versatile the category **Poly** really is.

## 4.1 Special polynomials and adjunctions

There are a few special classes of polynomials that are worth discussing:

- a) constant polynomials  $0, 1, 2, A$ ;
- b) linear polynomials  $0, y, 2y, Ay$ ;
- c) pure-power (or representable) polynomials  $1, y, y^2, y^A$ ; and
- d) monomials  $0, A, y, 2y^3, By^A$ .

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 4.2 and 4.3. The third puts a copy of **Set**<sup>op</sup> inside **Poly**: it is the Yoneda embedding that we saw way back in Exercise 2.13. Finally, the fourth puts a copy of bimorphic lenses inside **Poly**, as we saw in Section 2.3.3.

*Exercise 4.1.* 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 4.2.** There is a fully faithful functor  $\mathbf{Set} \rightarrow \mathbf{Poly}$  sending  $A \mapsto Ay^0 = A$ .

*Proof.* By (2.56), a map  $f: Ay^0 \rightarrow By^0$  consists of a function  $f: A \rightarrow B$  and, for each  $a \in A$ , a function  $0 \rightarrow 0$ . There is only one function  $0 \rightarrow 0$ , so  $f$  can be identified with just a map of sets  $A \rightarrow B$ .  $\square$

**Proposition 4.3.** There is a fully faithful functor  $\mathbf{Set} \rightarrow \mathbf{Poly}$  sending  $A \mapsto Ay$ .

*Proof.* By (2.56), a map  $f: Ay^1 \rightarrow By^1$  consists of a function  $f: A \rightarrow B$  and for each  $a \in A$  a function  $1 \rightarrow 1$ . There is only one function  $1 \rightarrow 1$ , so  $f$  can be identified with just a map of sets  $A \rightarrow B$ .  $\square$

**Theorem 4.4.**  $\mathbf{Poly}$  has an adjoint quadruple with  $\mathbf{Set}$ :

$$\begin{array}{ccc}
 & \xleftarrow{p(0)} & \\
 & \Rightarrow & \\
 \mathbf{Set} & \xrightarrow{A} & \mathbf{Poly} \\
 & \xleftarrow{p(1)} & \\
 & \Rightarrow & \\
 & \xrightarrow{Ay} & 
 \end{array} \tag{4.5}$$

where the functors have been labeled by where they send  $A \in \mathbf{Set}$  and  $p \in \mathbf{Poly}$ .

Both rightward functors  $\mathbf{Set} \rightarrow \mathbf{Poly}$  are fully faithful.

*Proof.* For any set  $A$ , there is a functor  $\mathbf{Poly} \rightarrow \mathbf{Set}$  given by sending  $p$  to  $p(A)$ ; by the Yoneda lemma, it is the functor  $\mathbf{Poly}(y^A, -)$ . This, together with Propositions 4.2 and 4.3, gives us the four functors and the fact that the two rightward functors are fully faithful. It remains to provide the following three natural isomorphisms:

$$\mathbf{Poly}(A, p) \cong \mathbf{Set}(A, p(0)) \quad \mathbf{Poly}(p, A) \cong \mathbf{Set}(p(1), A) \quad \mathbf{Poly}(Ay, p) \cong \mathbf{Set}(A, p(1)).$$

All three come from our formula (2.55) for computing general hom-sets in  $\mathbf{Poly}$ ; we leave the details to the reader in Exercise 4.6.  $\square$

*Exercise 4.6.* Here we prove the remainder of Theorem 4.4 using (2.55):

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  $\mathbf{Poly}(Ay, p) \cong \mathbf{Set}(A, p(1))$ .  $\diamond$

*Exercise 4.7.* Show that for any polynomial  $p$ , its set  $p(1)$  of positions is in bijection with the set of functions  $y \rightarrow p$ .  $\diamond$



In Theorem 4.4 we see that  $p \mapsto p(0)$  and  $p \mapsto p(1)$  have left adjoints. This is true more generally for any set  $A$  in place of 0 and 1, as we show in Corollary 4.10. However, the fact that  $p \mapsto p(1)$  is itself the left adjoint of the left adjoint of  $p \mapsto p(0)$ —and hence that we have the *quadruple* of adjunctions in (4.5)—is special to  $A = 0, 1$ .

We also have a copower-hom-power two-variable adjunction between **Poly**, **Set**, and **Poly**.

**Proposition 4.8.** There is a two-variable adjunction between **Poly**, **Set**, and **Poly**:

$$\mathbf{Poly}(Ap, q) \cong \mathbf{Set}(A, \mathbf{Poly}(p, q)) \cong \mathbf{Poly}(p, q^A). \quad (4.9)$$

*Proof.* Since  $Ap$  is the  $A$ -fold coproduct of  $p$  and  $q^A$  is the  $A$ -fold product of  $q$ , the universal properties of coproducts and products give natural isomorphisms

$$\mathbf{Poly}(Ap, q) \cong \prod_{a \in A} \mathbf{Poly}(p, q) \cong \mathbf{Poly}(p, q^A).$$

The middle set is naturally isomorphic to  $\mathbf{Set}(A, \mathbf{Poly}(p, q))$ , completing the proof.  $\square$

Replacing  $p$  with  $y^B$  in (4.9), we obtain the following using the Yoneda lemma.

**Corollary 4.10.** For any set  $B$  there is an adjunction

$$\mathbf{Set} \begin{array}{c} \xrightarrow{Ay^B} \\ \Rightarrow \\ \xleftarrow{q(B)} \end{array} \mathbf{Poly}$$

where the functors are labeled by where they send  $q \in \mathbf{Poly}$  and  $A \in \mathbf{Set}$ .

*Exercise 4.11.* Prove Corollary 4.10 from Proposition 4.8.  $\diamond$

**Proposition 4.12.** The Yoneda embedding  $A \mapsto y^A$  has a left adjoint

$$\mathbf{Set}^{\text{op}} \begin{array}{c} \xrightarrow{y^-} \\ \Leftarrow \\ \xleftarrow{\Gamma} \end{array} \mathbf{Poly}$$

where  $\Gamma(p) := \mathbf{Poly}(p, y) \cong \prod_{i \in p(1)} p[i]$ , as in (3.39) and (3.40). That is, there is a natural isomorphism

$$\mathbf{Poly}(p, y^A) \cong \mathbf{Set}(A, \Gamma(p)). \quad (4.13)$$

*Proof.* By (2.55), we have the natural isomorphism

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

which in turn is naturally isomorphic to  $\mathbf{Set}(A, \Gamma(p))$  by (3.40).  $\square$

**Corollary 4.14** (Principal monomial). There is an adjunction

$$\mathbf{Poly} \begin{array}{c} \xrightarrow{(p(1), \Gamma(p))} \\ \xRightarrow{\quad} \\ \xleftarrow{Ay^B} \end{array} \mathbf{Set} \times \mathbf{Set}^{\text{op}}$$

where the functors are labeled by where they send  $p \in \mathbf{Poly}$  and  $(A, B) \in \mathbf{Set} \times \mathbf{Set}^{\text{op}}$ . That is, there is a natural isomorphism

$$\mathbf{Poly}(p, Ay^B) \cong \mathbf{Set}(p(1), A) \times \mathbf{Set}(B, \Gamma(p)). \quad (4.15)$$

*Proof.* By the universal property of the product of  $A$  and  $y^B$ , we have a natural isomorphism

$$\mathbf{Poly}(p, Ay^B) \cong \mathbf{Poly}(p, A) \times \mathbf{Poly}(p, y^B).$$

Then the desired natural isomorphism follows from Exercise 4.6 #2 and (4.13).  $\square$

*Exercise 4.16.* Use (4.15) together with (3.85) and (3.89) to find an alternative proof for Proposition 3.56, i.e. that there is an isomorphism

$$\Gamma(p \otimes q) \cong \mathbf{Set}(q(1), \Gamma(p)) \times \mathbf{Set}(p(1), \Gamma(q)).$$

for any  $p, q \in \mathbf{Poly}$ .  $\diamond$

## 4.2 Epi-mono factorization of lenses

**Proposition 4.17.** Let  $f: p \rightarrow q$  be a lens in  $\mathbf{Poly}$ . It is a monomorphism if and only if the on-positions function  $f_1: p(1) \rightarrow q(1)$  is a monomorphism in  $\mathbf{Set}$  and, for each  $i \in p(1)$ , the on-directions function  $f_i^\sharp: q[f_1(i)] \rightarrow p[i]$  is an epimorphism in  $\mathbf{Set}$ .

*Proof.* To prove the forward direction, suppose that  $f$  is a monomorphism. Since  $p \mapsto p(1)$  is a right adjoint (Theorem 4.4), it preserves monomorphisms, so the on-positions function  $f_1$  is also a monomorphism.

We now need to show that for any  $i \in p(1)$ , the on-directions function  $f_i^\sharp: q[f_1(i)] \rightarrow p[i]$  is an epimorphism. Suppose we are given a set  $A$  and a pair of functions  $g^\sharp, h^\sharp: p[i] \rightrightarrows A$  with  $f_i^\sharp \circ g^\sharp = f_i^\sharp \circ h^\sharp$ . 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  $g \circ f = h \circ f$ . As  $f$  is a monomorphism,  $g = h$ ; in particular, their on-directions functions  $g^\sharp$  and  $h^\sharp$  are equal, as desired.

Conversely, suppose that  $f_1$  is a monomorphism and that, for each  $i \in p(1)$ , the function  $f_i^\sharp$  is an epimorphism. Let  $r$  be a polynomial and  $g, h: r \rightrightarrows p$  be two lenses such that  $g \circ f = h \circ f$ . Then  $g_1 \circ f_1 = h_1 \circ f_1$ , which implies  $g_1 = h_1$ ; we'll consider  $g_1$

the default representation. We also have that  $f_{g_1(k)}^\# \circ g_k^\# = f_{g_1(k)}^\# \circ h_k^\#$  for any  $k \in r(1)$ . But  $f_{g_1(k)}^\#$  is an epimorphism, so in fact  $g_k^\# = h_k^\#$ , as desired.  $\square$

*Example 4.18.* Choose a finite nonempty set  $k$  for  $1 \leq k \in \mathbb{N}$ , e.g.  $k = 12$ . There is a monomorphism

$$f: ky^k \rightarrow \mathbb{N}y^{\mathbb{N}}$$

such that the trajectory “going around and around the  $k$ -clock” comes from the usual counting trajectory  $\mathbb{N}y^{\mathbb{N}} \rightarrow y$  from Example 3.5.

On positions, we have  $f_1(i) = i$  for all  $i \in k$ . On directions, for any  $i \in k$ , we have  $f_i^\#(n) = n \bmod k$  for all  $n \in \mathbb{N}$ .

*Exercise 4.19.* In Example 4.18, we gave a map  $12y^{12} \rightarrow \mathbb{N}y^{\mathbb{N}}$ . 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  $12y^{12} \rightarrow p$  would be seen to relate to the behavior of the old system  $\mathbb{N}y^{\mathbb{N}} \rightarrow p$ .  $\diamond$

**Proposition 4.20.** Let  $f: p \rightarrow q$  be a lens in **Poly**. It is an epimorphism if and only if the function  $f_1: p(1) \rightarrow q(1)$  is an epimorphism in **Set** and, for each  $j \in q(1)$ , the induced function

$$f_j^b: q[j] \rightarrow \prod_{\substack{i \in p(1), \\ f_1(i)=j}} p[i]$$

from (2.63) is a monomorphism.

*Proof.* To prove the forward direction, suppose that  $f$  is an epimorphism. Since  $p \mapsto p(1)$  is a left adjoint (Theorem 4.4), it preserves epimorphisms, so the on-positions function  $f_1$  is also a epimorphism.

We now need to show that for any  $j \in q(1)$ , the induced function  $f_j^b$  is a monomorphism. Suppose we are given a set  $A$  and a pair of functions  $g', h': A \rightrightarrows q[j]$  with  $g' \circ f_j^b = h' \circ f_j^b$ . They can be identified with lenses  $g, h: q \rightrightarrows y^A + 1$ , 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  $g^\# = h^\#$  as desired.

Then we can construct lenses  $g, h: q \rightrightarrows y^A + 1$  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  $g_j^\# := g'$  and  $h_j^\# := h'$ . Then  $f \circ g = f \circ h$ . 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  $f_j^b$  is a monomorphism. Let  $r$  be a polynomial and  $g, h: q \rightrightarrows r$  be two lenses such that  $f \circ g = f \circ h$ . Then  $f_1 \circ g_1 = f_1 \circ h_1$ , which implies  $g_1 = h_1$ ; we'll consider  $g_1$  the default representation. We also have that  $g_{f_1(i)}^\# \circ f_i^\# = h_{f_1(i)}^\# \circ f_i^\#$  for any  $i \in p(1)$ . It follows that, for any  $j \in q(1)$ , the two composites

$$r[g_1(j)] \xrightarrow[h_j^\#]{g_j^\#} q[j] \xrightarrow{f_j^b} \prod_{\substack{i \in p(1), \\ f_1(i)=j}} p[i]$$

are equal, which implies that  $g_j^\# = h_j^\#$  as desired.  $\square$

*Exercise 4.21.* Show that the only way for a map  $p \rightarrow y$  to *not* be an epimorphism is when  $p = 0$ .  $\diamond$

*Exercise 4.22.* Let  $A$  and  $B$  be sets and  $AB$  their product. Find an epimorphism  $y^A + y^B \twoheadrightarrow y^{AB}$ .  $\diamond$

*Exercise 4.23.* Suppose a lens  $f: p \rightarrow q$  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.

$\diamond$

We are often interested in whether epimorphisms and monomorphisms form what is called a *factorization system* in a given category, which we define below.

**Definition 4.24** (Factorization system). Given a category  $\mathcal{C}$  and two classes of morphisms  $E$  and  $M$  in  $\mathcal{C}$ , we say that  $(E, M)$  is a *factorization system* of  $\mathcal{C}$  if:

1. every morphism  $f$  in  $\mathcal{C}$  factors uniquely (up to unique isomorphism) as a morphism  $e \in E$  composed with a morphism  $m \in M$ , so that  $f = e \circ m$ ;
2.  $E$  and  $M$  each contain every isomorphism; and
3.  $E$  and  $M$  are each closed under composition.

If  $E$  is the class of epimorphisms and  $M$  is the class of monomorphisms (in which case conditions 2 and 3 are automatically satisfied), we say that  $\mathcal{C}$  has *epi-mono factorization*.

*Example 4.25* (Epi-mono factorization in **Set**). The category **Set** has epi-mono factorization: a function  $f: X \rightarrow Y$  can be uniquely factored into an epimorphism (surjection)  $e$  followed by a monomorphism (injection)  $i$ , as follows. The epimorphism  $e: X \rightarrow f(X)$  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  $i: f(X) \rightarrow Y$  is then given by including the image into the codomain, so  $i$  sends  $y \mapsto y$  for all  $y \in f(X) \subseteq Y$ .

**Proposition 4.26.** **Poly** has epi-mono factorization.

*Proof.* Take an arbitrary lens  $\varphi: p \rightarrow q$ . It suffices to show that there exists a unique polynomial  $r$  equipped with an epimorphism  $\epsilon: p \rightarrow r$  and a monomorphism  $\mu: r \rightarrow q$  such that  $\varphi = \epsilon \circ \mu$ .

On positions, we must have  $\varphi_1 = \epsilon_1 \circ \mu_1$ , with  $\mu_1$  a monomorphism and  $\epsilon_1$  an epimorphism per Propositions 4.17 and 4.20. By Example 4.25, since **Set** has epi-mono factorization, such  $r(1)$ ,  $\epsilon_1$ , and  $\mu_1$  uniquely exist. In particular, we must have that  $r(1) \cong \varphi_1(p(1))$ , that  $\epsilon_1: p(1) \rightarrow \varphi_1(p(1))$  is the corestriction of  $\varphi_1$  sending  $i \mapsto \varphi_1(i)$  for each  $p$ -position  $i$ , and that  $\mu_1: \varphi_1(p(1)) \rightarrow q(1)$  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

$$\begin{array}{ccc} q[\varphi_1(i)] & \xrightarrow{\mu_{\varphi_1(i)}^\#} & r[\varphi_1(i)] \\ & \searrow \varphi_i^\# & \downarrow \epsilon_i^\# \\ & & p[i] \end{array}$$

commutes—or, equivalently, for every  $j \in r(1) \cong \varphi_1(p(1))$ ,

$$\begin{array}{ccc} q[j] & \xrightarrow{\mu_j^\#} & r[j] \\ & \searrow \varphi_j^b & \downarrow \epsilon_j^b \\ & & \prod_{\substack{i \in p(1), \\ \varphi_1(i)=j}} p[i] \end{array}$$

commutes (here  $\varphi_j^b$  and  $\epsilon_j^b$  are the induced functions from (2.63)), with  $\mu_j^\#$  an epimorphism and  $\epsilon_j^b$  a monomorphism per Propositions 4.17 and 4.20. So again since **Set** has epi-mono factorization, such  $r[j]$ ,  $\mu_j^\#$ , and  $\epsilon_j^b$  uniquely exist. Hence such  $p \xrightarrow{\epsilon} r \xrightarrow{\mu} q$  uniquely exists overall.  $\square$

### 4.3 Cartesian closure

We have already seen the closure operation  $[-, -]$  for one monoidal structure on **Poly**, namely  $(y, \otimes)$ . But this is not the only closed monoidal structure on **Poly**: in fact, we will show that **Poly** is cartesian closed as well.

For any two polynomials  $q, r$ , define  $r^q \in \mathbf{Poly}$  by the formula

$$r^q := \prod_{j \in q(1)} r \circ (y + q[j]) \quad (4.27)$$

where  $\circ$  denotes composition.

Before proving that this really is an exponential in **Poly**, which we do in Theorem 4.30, we first get some practice with it.

*Example 4.28.* Let  $A$  be a set. We've been writing the polynomial  $Ay^0$  simply as  $A$ , so it better be true that there is an isomorphism

$$y^A \cong y^{Ay^0}$$

in order for the notation to be consistent. Luckily, this is true. By (4.27), we have

$$y^{Ay^0} = \prod_{a \in A} y \circ (y + 0) \cong y^A$$

*Exercise 4.29.* Compute the following exponentials in **Poly** using (4.27):

1.  $p^0$  for an arbitrary  $p \in \mathbf{Poly}$ .
2.  $p^1$  for an arbitrary  $p \in \mathbf{Poly}$ .
3.  $1^p$  for an arbitrary  $p \in \mathbf{Poly}$ .
4.  $A^p$  for an arbitrary  $p \in \mathbf{Poly}$  and  $A \in \mathbf{Set}$ .
5.  $y^y$ .
6.  $y^{4y}$ .
7.  $(y^A)^{y^B}$  for arbitrary sets  $A, B \in \mathbf{Set}$ . ◇

**Theorem 4.30.** The category **Poly** is cartesian closed. That is, we have a natural isomorphism

$$\mathbf{Poly}(p, r^q) \cong \mathbf{Poly}(p \times q, r),$$

where  $r^q$  is the polynomial defined in (4.27).

*Proof.* We have the following chain of natural isomorphisms:

$$\mathbf{Poly}(p, r^q) \cong \mathbf{Poly}\left(p, \prod_{j \in q(1)} r \circ (y + q[j])\right) \quad (4.27)$$

$$\begin{aligned}
&\cong \prod_{i \in p(1)} \prod_{j \in q(1)} \mathbf{Poly}(y^{p[i]}, r \circ (y + q[j])) \\
&\quad \text{(Universal property of (co)products)} \\
&\cong \prod_{i \in p(1)} \prod_{j \in q(1)} r \circ (p[i] + q[j]) \quad \text{(Yoneda lemma)} \\
&\cong \prod_{i \in p(1)} \prod_{j \in q(1)} \sum_{k \in r(1)} (p[i] + q[j])^{r[k]} \\
&\cong \prod_{(i,j) \in (p \times q)(1)} \sum_{k \in r(1)} (p \times q)[(i, j)]^{r[k]} \quad (2.85) \\
&\cong \mathbf{Poly}(p \times q, r). \quad (2.55)
\end{aligned}$$

□

*Exercise 4.31.* Use Theorem 4.30 to show that for any polynomials  $p, q$ , there is a canonical evaluation map

$$\text{eval}: p^q \times q \rightarrow p.$$

◇

## 4.4 Limits and colimits of polynomials

We have already seen that **Poly** has all coproducts (Proposition 2.51) and products (Proposition 2.80). We will now see that **Poly** has all small limits and colimits.

**Theorem 4.32.** The category **Poly** has all small limits.

*Proof.* A category has all small limits if and only if it has products and equalizers, so by Proposition 2.80, 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  $f, g: p \rightrightarrows q$  be two lenses. We construct the equalizer  $p'$  of  $f$  and  $g$  as follows.<sup>1</sup> We define its position-set  $p'(1)$  to be the equalizer of  $f_1, g_1: p(1) \rightrightarrows q(1)$  in **Set**; that is,

$$p'(1) := \{i \in p(1) \mid f_1(i) = g_1(i)\}.$$

Then for each  $i \in p'(1)$ , we can define the direction-set  $p'[i]$  to be the coequalizer of  $f_i^\#, g_i^\#: q[f_1(i)] \rightrightarrows p[i]$ . In this way, we obtain a polynomial  $p'$  that comes equipped with a lens  $e: p' \rightarrow p$ . One can check that  $p'$  together with  $e$  satisfies the universal property of the equalizer of  $f$  and  $g$ ; see Exercise 4.33. □

<sup>1</sup>If we're being precise, a "(co)equalizer" is an object equipped with a map, but we will use the term to refer to either just the object or just the map when the context is clear.

*Exercise 4.33.* Complete the proof of Theorem 4.32 as follows:

1. We said that  $p'$  comes equipped with a lens  $e: p' \rightarrow p$ ; what is it?
2. Show that  $e \circ f = e \circ g$ .
3. Show that  $e$  is the equalizer of the pair  $f, g$ . ◇

*Example 4.34* (Computing general limits in **Poly**). The proof of Theorem 4.32 justifies the following mnemonic for limits in **Poly**:

*The positions of a limit are the limit of the positions.  
The directions of a limit are the colimit of the directions.*

We can make this precise as follows: the limit of a functor  $p_-: \mathcal{G} \rightarrow \mathbf{Poly}$  is the polynomial whose position-set is

$$\left( \lim_{j \in \mathcal{G}} p_j \right) (1) \cong \lim_{j \in \mathcal{G}} p_j(1), \quad (4.35)$$

equipped with a canonical projection  $\pi_j$  to each  $p_j(1)$ , and whose direction-set for each position  $i$  is

$$\left( \lim_{j \in \mathcal{G}} p_j \right) [i] \cong \operatorname{colim}_{j \in \mathcal{G}^{\text{op}}} p_j[\pi_j(i)]. \quad (4.36)$$

This notation obscures what is occurring on lenses, but in particular, each lens  $\varphi: p_j \rightarrow p_{j'}$  in the diagram  $p_-$  induces an on-positions function  $\varphi_1: p_j(1) \rightarrow p_{j'}(1)$  in the diagram whose limit we take in (4.35) and, for every position  $i$  of the limit, an on-directions function  $\varphi_{\pi_j(i)}^\# : p_{j'}[\pi_{j'}(i)] \rightarrow p_j[\pi_j(i)]$  in the diagram whose colimit we take in (4.36). (Note that, by the definition of a limit,  $\varphi_1(\pi_j(i)) = \pi_{j'}(i)$ .)

We have seen (4.35) and (4.36) to be true for products: the position-set of the product is just the product of the original position-sets, while the direction-set at a tuple of the original positions is just the coproduct of the direction-sets at every position in the tuple. We have also just shown (4.35) and (4.36) to be true for equalizers in the proof of Theorem 4.32. It follows from the construction of any limit as an equalizer of products that it is true for arbitrary limits.

*Example 4.37* (Pullbacks in **Poly**). Given  $q, q', r \in \mathbf{Poly}$  and lenses  $q \xrightarrow{f} r \xleftarrow{f'} q'$ , the pullback

$$\begin{array}{ccc} p & \xrightarrow{g'} & q' \\ g \downarrow & \lrcorner & \downarrow f' \\ q & \xrightarrow{f} & r \end{array}$$

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:

$$\begin{array}{ccc} p(1) & \xrightarrow{g'_1} & q'(1) \\ g_1 \downarrow & \lrcorner & \downarrow f'_1 \\ q(1) & \xrightarrow{f_1} & r(1) \end{array} \quad \text{and} \quad \begin{array}{ccc} p[(i, i')] & \xleftarrow{(g')^\#_{(i, i')}} & q'[i'] \\ g^\#_{(i, i')} \uparrow & \lrcorner & \uparrow (f')^\#_{i'} \\ q[i] & \xleftarrow{f_i^\#} & r[f_1(i)] \end{array} \quad (4.38)$$

*Exercise 4.39.* Let  $p$  be any polynomial.

1. There is a canonical choice of lens  $\eta: p \rightarrow p(1)$ ; what is it?
2. Given an element  $i \in p(1)$ , i.e. a function (or lens between constant polynomials)  $i: 1 \rightarrow p(1)$ , let  $p_i$  be the pullback

$$\begin{array}{ccc} p_i & \xrightarrow{g} & p \\ f \downarrow & \lrcorner & \downarrow \eta \\ 1 & \xrightarrow{i} & p(1) \end{array}$$

What is  $p_i$ ? What are the maps  $f: p_i \rightarrow 1$  and  $g: p_i \rightarrow p$ ? ◇

*Exercise 4.40.* Let  $q := y^2 + y$ ,  $q' := 2y^3 + y^2$ , and  $r := y + 1$ .

1. Choose lenses  $f: q \rightarrow r$  and  $f': q' \rightarrow r$  and write them down.
2. Find the pullback of  $q \xrightarrow{f} r \xleftarrow{f'} q'$ . ◇

*Exercise 4.41.* An alternative way to prove Theorem 4.32 would have been to show that the equalizer of two natural transformations between polynomial functors in  $\mathbf{Set}^{\mathbf{Set}}$  is still a polynomial functor—since the full subcategory inclusion  $\mathbf{Poly} \rightarrow \mathbf{Set}^{\mathbf{Set}}$  reflects these equalizers, it would follow that  $\mathbf{Poly}$  has equalizers. But we already know what polynomial the equalizer should be from the proof of Theorem 4.32. So in this exercise, we will show that the equalizer of polynomials we found in  $\mathbf{Poly}$  is also the equalizer of those same functors in  $\mathbf{Set}^{\mathbf{Set}}$ .

Let  $f, g: p \rightrightarrows q$  be a pair of natural transformations  $f, g: p \rightrightarrows q$  between polynomial functors  $p$  and  $q$ , and let  $e: p' \rightarrow p$  be their equalizer in  $\mathbf{Poly}$  that we computed in the proof of Theorem 4.32.

1. Given a set  $X$ , show that  $e_X: p'(X) \rightarrow p(X)$  is the equalizer of the  $X$ -components  $f_X, g_X: p(X) \rightrightarrows q(X)$  in **Set**.
2. Deduce that equalizers in  $\mathbf{Poly}$  coincide with equalizers in  $\mathbf{Set}^{\mathbf{Set}}$ .

3. Conclude that limits in **Poly** coincide with limits in  $\mathbf{Set}^{\mathbf{Set}}$ .  $\diamond$

**Theorem 4.42.** The category **Poly** has all small colimits.

*Proof.* A category has all small colimits if and only if it has coproducts and coequalizers, so by Proposition 2.51, it suffices to show that **Poly** has coequalizers.

Let  $s, t: p \rightrightarrows q$  be two lenses. We construct the coequalizer  $q'$  of  $s$  and  $t$  as follows. The pair of functions  $s_1, t_1: p(1) \rightrightarrows q(1)$  define a graph  $G: \bullet \rightrightarrows \bullet \rightarrow \mathbf{Set}$  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  $g_1: q(1) \rightarrow C$  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  $V_c := g_1^{-1}(c)$  and edges  $E_c := s_1^{-1}(g_1^{-1}(c)) = t_1^{-1}(g_1^{-1}(c))$ . 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,  $e \rightarrow s_1(e)$  and  $e \rightarrow t_1(e)$ , 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  $F: (\int G_c)^{\text{op}} \rightarrow \mathbf{Set}$  sending every  $v \mapsto q[v]$ , every  $e \mapsto p[e]$ , and every morphism to a function between them, namely either  $s_e^\sharp: q[s_1(e)] \rightarrow p[e]$  or  $t_e^\sharp: q[t_1(e)] \rightarrow p[e]$ . So we can define  $q'[c]$  to be the limit of  $F$  in **Set**.

We claim that  $q' := \sum_{c \in C} y^{q'[c]}$  is the coequalizer of  $s$  and  $t$ . We leave the complete proof to the interested reader in Exercise 4.43.  $\square$

*Exercise 4.43.* Complete the proof of Theorem 4.42 as follows:

1. Provide a map  $g: q \rightarrow q'$ .
2. Show that  $s \circ g = t \circ g$ .
3. Show that  $g$  is a coequalizer of the pair  $s, t$ .  $\diamond$

*Example 4.44.* Given a diagram in **Poly**, one could either take its (co)limit as a diagram of *polynomial* functors (i.e. its (co)limit in **Poly**) or its (co)limit simply as a diagram of functors (i.e. its (co)limit in  $\mathbf{Set}^{\mathbf{Set}}$ ). We saw in Exercise 4.41 that in the case of limits, these yield the same result. So, too, in the case of coproducts, per Proposition 2.51.

But in the case of general colimits, there are diagrams that yield different results: by the co-Yoneda lemma, *every* functor  $\mathbf{Set} \rightarrow \mathbf{Set}$ —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  $y^2 \rightarrow y$ , which form the diagram

$$y^2 \rightrightarrows y. \quad (4.45)$$

According to Theorem 4.42, the colimit of (4.45) in **Poly** has the coequalizer of  $1 \rightrightarrows 1$ , namely 1, as its position-set, and the limit of the diagram  $1 \rightrightarrows 2$  consisting of the two inclusions as its sole direction-set. But this latter limit is just 0, so in fact the colimit of (4.45) in **Poly** is the constant functor  $1y^0 \cong 1$ .

But as functors, by Proposition 2.34, the colimit of (4.45) can be computed pointwise: it is the (nonconstant!) functor

$$X \mapsto \begin{cases} 0 & \text{if } X = 0 \\ 1 & \text{if } X \neq 0 \end{cases}$$

*Exercise 4.46.* By Proposition 1.18, for any polynomial  $p$ , there are canonical maps

$$\epsilon: p(1)y \rightarrow p \quad \text{and} \quad \eta: p \rightarrow y^{\Gamma(p)}.$$

1. Characterize the behavior of the canonical map  $\epsilon: p(1)y \rightarrow p$ .
2. Characterize the behavior of the canonical map  $\eta: p \rightarrow y^{\Gamma(p)}$ .
3. Show that the following is a pushout in **Poly**:

$$\begin{array}{ccc} p(1)y & \xrightarrow{!} & y \\ \epsilon \downarrow & \lrcorner & \downarrow ! \\ p & \xrightarrow{\eta} & y^{\Gamma(p)} \end{array} \quad (4.47)$$

◇

**Proposition 4.48.** For polynomials  $p, q$ , the following is a pushout:

$$\begin{array}{ccc} p(1)y \otimes q(1)y & \longrightarrow & p(1)y \otimes q \\ \downarrow & \lrcorner & \downarrow \\ p \otimes q(1)y & \longrightarrow & p \otimes q \end{array}$$

*Proof.* All the maps shown are identities on positions, so the displayed diagram is the coproduct over all  $(i, j) \in p(1) \times q(1)$  of the diagram shown left

$$\begin{array}{ccccc} y & \longrightarrow & y^{q[j]} & 1 & \longleftarrow & q[j] \\ \downarrow & & \downarrow & \uparrow & & \uparrow \\ y^{p[i]} & \longrightarrow & y^{p[i] \times q[j]} & p[i] & \longleftarrow & p[i] \times q[j] \end{array}$$

where we used  $(p \otimes q)[(i, j)] \cong p[i] \times q[j]$ . This is the image under the Yoneda embedding of the diagram of sets shown right, which is clearly a pullback. The result follows by Proposition 4.12.  $\square$

This means that to give a map  $\varphi: p \otimes q \rightarrow r$ , it suffices to give two maps  $\varphi_p: p \otimes q(1)y \rightarrow r$  and  $\varphi_q: p(1)y \otimes q \rightarrow r$  that agree on positions. The map  $\varphi_p$  says how information about  $q$ 's position is transferred to  $p$ , and the map  $\varphi_q$  says how information about  $p$ 's position is transferred to  $q$ .

**Corollary 4.49.** Suppose we have polynomials  $p_1, \dots, p_n \in \mathbf{Poly}$ . Then  $p_1 \otimes \dots \otimes p_n$  is isomorphic to the wide pushout

$$\text{colim} \left( \begin{array}{ccc} & p_1(1)y \otimes \dots \otimes p_n(1)y & \\ \swarrow & & \searrow \\ p_1 \otimes p_2(1)y \otimes \dots \otimes p_n(1)y & \dots & p_1(1)y \otimes \dots \otimes p_{n-1}(1)y \otimes p_n \end{array} \right)$$

*Proof.* We proceed by induction on  $n \in \mathbb{N}$ . When  $n = 0$ , the wide pushout has no legs and the empty parallel product is  $y$ , so the result holds. If the result holds for  $n$ , then it holds for  $n + 1$  by Proposition 4.48.  $\square$

## 4.5 Vertical-cartesian factorization of lenses

Aside from epi-mono factorization, there is another factorization system on  $\mathbf{Poly}$  that will show up frequently.

**Definition 4.50** (Vertical and cartesian lenses). Let  $f: p \rightarrow q$  be a lens. It is called *vertical* if  $f_1: p(1) \rightarrow q(1)$  is an isomorphism. It is called *cartesian* if, for each  $i \in p(1)$ , the function  $f_i^\sharp: q[f(i)] \rightarrow p[i]$  is an isomorphism.

**Proposition 4.51.** Vertical and cartesian lenses form a factorization system of  $\mathbf{Poly}$ .

*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  $\mathbf{Poly}$  can be uniquely (up to unique isomorphism) factored as a vertical lens composed with a cartesian lens.

Recall from (2.56) that a lens in  $\mathbf{Poly}$  can be written as to the left; we can thus rewrite it as to the right:

$$\begin{array}{ccc} p(1) & \xrightarrow{f_1} & q(1) \\ & \Downarrow f^\sharp & \\ p[-] & \xrightarrow{\quad} & \mathbf{Set} \end{array} \quad \begin{array}{ccc} p(1) & \xrightarrow{\quad} & p(1) \xrightarrow{f_1} q(1) \\ & \Downarrow f^\sharp & \downarrow q[f_1(-)] \\ p[-] & \xrightarrow{\quad} & \mathbf{Set} \end{array}$$

We can see that the intermediary object  $\sum_{i \in p(1)} y^{q[f_1(i)]}$  is unique up to unique isomorphism.  $\square$

**Proposition 4.52.** Vertical lenses satisfy 2-out-of-3: given  $p \xrightarrow{f} q \xrightarrow{g} r$  with  $h = f \circ g$ , 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  $h = f \circ g$ , we have that  $h_1 = f_1 \circ g_1$ . 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,  $h = f \circ g$  implies that for every  $i \in p(1)$ , we have  $h_i^\# = g_{f_1(i)}^\# \circ f_i^\#$ . Since  $g_{f_1(i)}^\#$  is an isomorphism, it follows that every  $h_i^\#$  is an isomorphism if and only if every  $f_i^\#$  is an isomorphism, so  $h$  is cartesian if and only if  $f$  is cartesian.  $\square$

*Exercise 4.53.* Give an example of polynomials  $p, q, r$  and maps  $p \xrightarrow{f} q \xrightarrow{g} r$  such that  $f$  and  $f \circ g$  are cartesian but  $g$  is not.  $\diamond$

Here is an alternative characterization of a cartesian lens in **Poly**. Recall from Exercise 2.66 that for any polynomial  $p$ , there is a corresponding function  $\pi_p: \dot{p}(1) \rightarrow p(1)$ , i.e. the set of all directions mapping to the set of positions. A lens  $(f_1, f^\#): p \rightarrow q$  can then be described as a function  $f_1: p(1) \rightarrow q(1)$  along with a function  $f^\#$  that makes the following diagram in **Set** commute:

$$\begin{array}{ccccc}
 \dot{p}(1) & \xleftarrow{f^\#} & \bullet & \xrightarrow{\quad} & \dot{q}(1) \\
 \pi_p \downarrow & & \downarrow & \lrcorner & \downarrow \pi_q \\
 p(1) & \xlongequal{\quad} & p(1) & \xrightarrow{f_1} & q(1)
 \end{array} \tag{4.54}$$

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  $f^\#$  sends each such pair to a direction  $f_i^\#(e)$  of  $p$ , and the commutativity of the left square implies that  $f_i^\#(e)$  is specifically a  $p[i]$ -direction. So  $f_i^\#$  is indeed our familiar on-directions function  $q[f_1(i)] \rightarrow p[i]$ , and  $f^\#$  is just the sum of all these on-directions functions over  $i \in p(1)$ .

*Exercise 4.55.* Show that a lens  $f: p \rightarrow q$  in **Poly** is cartesian if and only if the square on

the left hand side of (4.54) is also a pullback:

$$\begin{array}{ccccc}
 \dot{p}(1) & \xleftarrow{f^\#} & \bullet & \xrightarrow{\quad} & \dot{q}(1) \\
 \pi_p \downarrow & & \downarrow & \lrcorner & \downarrow \pi_q \\
 p(1) & \xlongequal{\quad} & p(1) & \xrightarrow{f_1} & q(1)
 \end{array}$$

◇

*Exercise 4.56.* Is the pushout of a cartesian map always cartesian? ◇

Why do we use the word *cartesian* to describe cartesian morphisms? It turns out that, as natural transformations, cartesian morphisms are precisely what are known as cartesian natural transformations.

**Definition 4.57** (Cartesian natural transformation). A *cartesian natural transformation* is a natural transformation whose naturality squares are all pullbacks. That is, given categories  $\mathcal{C}, \mathcal{D}$ , functors  $F, G$ , and natural transformation  $\alpha$ , we say that  $\alpha$  is *cartesian* if for all morphisms  $h: c \rightarrow c'$  in  $\mathcal{C}$ ,

$$\begin{array}{ccc}
 Fc & \xrightarrow{\alpha_c} & Gc \\
 Fh \downarrow & \lrcorner & \downarrow Gh \\
 Fd & \xrightarrow{\alpha_d} & Gd
 \end{array}$$

is a pullback.

**Proposition 4.58.** Let  $f: p \rightarrow q$  be a morphism in **Poly**. The following are equivalent:

1. viewed as a lens,  $f$  is cartesian in the sense of Definition 4.50: for each  $i \in p(1)$ , the on-directions function  $f_i^\#$  is a bijection;
2. the square on the left hand side of (4.54) is also a pullback:

$$\begin{array}{ccccc} p(1) & \xleftarrow{f^\#} & \bullet & \xrightarrow{\quad} & q(1) \\ \pi_p \downarrow & & \downarrow & \lrcorner & \downarrow \pi_q \\ p(1) & \xlongequal{\quad} & p(1) & \xrightarrow{f_1} & q(1) \end{array}$$

3. viewed as a natural transformation,  $f$  is cartesian in the sense of Definition 4.57: for any sets  $A, B$  and function  $h: A \rightarrow B$ , the naturality square

$$\begin{array}{ccc} p(A) & \xrightarrow{f_A} & q(A) \\ p(h) \downarrow & \lrcorner & \downarrow q(h) \\ p(B) & \xrightarrow{f_B} & q(B) \end{array} \quad (4.59)$$

is a pullback.

*Proof.* We already showed that the first two are equivalent in Exercise 4.55, and we will complete this proof in Exercise 4.60.  $\square$

*Exercise 4.60.* In this exercise, you will complete the proof of Proposition 4.58.

First, we will show that  $1 \Rightarrow 3$ . In the following, let  $f: p \rightarrow q$  be a cartesian lens in **Poly** and  $h: A \rightarrow B$  be a function.

1. Using Proposition 2.71 to translate  $f$  from a lens in **Poly** to a natural transformation and Proposition 2.47 to interpret  $q(h)$ , characterize the pullback of  $p(B) \xrightarrow{f_B} q(B) \xleftarrow{q(h)} q(A)$  in **Set**.
2. Show that this pullback coincides with the naturality square (4.59), hence proving  $1 \Rightarrow 3$ .

Next, we show that  $3 \Rightarrow 1$ . In the following, let  $f: p \rightarrow q$  be a lens in **Poly** that is cartesian when viewed as a natural transformation, so that (4.59) is a pullback for any function  $h: A \rightarrow B$ . Also fix  $i \in p(1)$ .

3. Show that the diagram

$$\begin{array}{ccc} 1 & \xrightarrow{(f_1(i), \text{id}_{q[f_1(i)]})} & q(q[f_1(i)]) \\ (i, \text{id}_{p[i]}) \downarrow & \lrcorner & \downarrow q(f_i^\#) \\ p(p[i]) & \xrightarrow{f_{p[i]}} & q(p[i]) \end{array} \quad (4.61)$$

commutes. Hint: Use Proposition 2.47, Proposition 2.71, and/or Corollary 2.73.

4. Apply the universal property of the pullback (4.59) to the diagram (4.61) above to exhibit an element of  $p(q[f_1(i)])$ . Conclude from the existence of this element that  $f_i^\#$  is an isomorphism, hence proving  $3 \Rightarrow 1$ .  $\diamond$

**Proposition 4.62.** The monoidal structures  $+$ ,  $\times$ , and  $\otimes$  preserve both vertical and cartesian morphisms.

*Proof.* Suppose that  $f: p \rightarrow p'$  and  $g: q \rightarrow q'$  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 4.4. As this functor is both a left adjoint and a right adjoint, it preserves both sums and products, so  $(f + g)_1 = f_1 + g_1$  and  $(f \times g)_1 = f_1 \times g_1$ . Hence  $f + g$  and  $f \times g$  are both vertical. On-positions, the behavior of  $\otimes$  is identical to the behavior of  $\times$ , so  $f \otimes g$  must be vertical as well.

Now suppose that  $f: p \rightarrow p'$  and  $g: q \rightarrow q'$  are cartesian.

A position of  $p + q$  is a position  $i \in p(1)$  or a position  $j \in q(1)$ , and the map  $(f + g)^\#$  at that position is either  $f_i^\#$  or  $g_j^\#$ ; either way it is an isomorphism, so  $f + g$  is cartesian.

A position of  $p \times q$  (resp. of  $p \otimes q$ ) is a pair  $(i, j) \in p(1) \times q(1)$ . The lens  $(f \times g)^\#_{(i,j)}$  (resp.  $(f \otimes g)^\#_{(i,j)}$ ) is  $f_i^\# + g_j^\#$  (resp.  $f_i^\# \times g_j^\#$ ) which is again an isomorphism if  $f_i^\#$  and  $g_j^\#$  are. Hence  $f \times g$  (resp.  $f \otimes g$ ) is cartesian, completing the proof.  $\square$

**Proposition 4.63.** Pullbacks preserve vertical (resp. cartesian) lenses. In other words, if  $f: p \rightarrow q$  is a lens and  $g: q' \rightarrow q$  a vertical (resp. cartesian) lens, then the pullback  $g'$  of  $g$  along  $p$

$$\begin{array}{ccc} p \times_q q' & \longrightarrow & q' \\ g' \downarrow & \lrcorner & \downarrow g \\ p & \xrightarrow{f} & q \end{array}$$

is vertical (resp. cartesian).

*Proof.* This follows from Example 4.37, since the pullback (resp. pushout) of an isomorphism is an isomorphism.  $\square$

## 4.6 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  $\mathbf{Set}$  acts as a sort of remote controller on the category  $\mathbf{Poly}$ , 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  $f: A \rightarrow p(1)$ , 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

$$\begin{array}{ccc} f^*p & \xrightarrow{\text{cart}} & p \\ \downarrow & \lrcorner & \downarrow \eta_p \\ A & \xrightarrow{f} & p(1) \end{array} \quad (4.64)$$

Here  $\eta_p$  is the unit of the adjunction  $\mathbf{Set} \xrightleftharpoons[p(1)]{A} \mathbf{Poly}$ ; it is a vertical lens. We could evaluate this pullback using Example 4.37. Alternatively, we can use Proposition 4.63 to deduce that the top map  $f^*p \rightarrow p$  (which we presciently labeled  $\text{cart}$ ) is cartesian like  $f$  and that the left map  $f^*p \rightarrow A$  is vertical like  $\eta_p$ . Furthermore,  $\text{cart}_1 = f$ . Hence

$$f^*p \cong \sum_{a \in A} y^{p[f(a)]}.$$

We'll see this as part of a bigger picture in Proposition 4.71 and Theorem 4.72, but first we need the following definitions and a result about cartesian lenses.

**Definition 4.65** (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 4.66** (Exponentiable morphism). Given a category  $\mathcal{C}$  with objects  $c, d$  and morphism  $f: c \rightarrow d$  such that all pullbacks along  $f$  exist in  $\mathcal{C}$ , we say that  $f$  is *exponentiable* if the functor  $f^*: \mathcal{C}/d \rightarrow \mathcal{C}/c$  given by pulling back along  $f$  is a left adjoint.

**Theorem 4.67.** Cartesian lenses in **Poly** are exponentiable. That is, if  $f: p \rightarrow q$  is cartesian, then the functor  $f^*: \mathbf{Poly}/q \rightarrow \mathbf{Poly}/p$  given by pulling back along  $f$  is a left adjoint:

$$\mathbf{Poly}/p \xrightleftharpoons[f_*]{f^*} \mathbf{Poly}/q$$

*Proof.* Fix  $e: p' \rightarrow p$  and  $g: q' \rightarrow q$ .

$$\begin{array}{ccc} p' & & q' \\ e \downarrow & & \downarrow g \\ p & \xrightarrow{f} & q \end{array}$$

We need to define a functor  $f_*: \mathbf{Poly}/p \rightarrow \mathbf{Poly}/q$  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  $Q \rightarrow P'_i$ , 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:

$$f_*p' := \sum_{j \in q(1)} \sum_{i' \in \prod_{i \in f_1^{-1}(j)} e_1^{-1}(i)} y^{q[j]/\sum_{i \in f_1^{-1}(j)} p'[i'(i)]} \quad (4.68)$$

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

$$\begin{aligned} (\mathbf{Poly}/p)(f^*q', p') &\cong \prod_{i \in p(1)} \prod_{\{j' \in q'(1) \mid g_1(j') = f_1(i)\}} \sum_{\{i' \in p'(1) \mid e_1(i') = i\}} (p[i]/\mathbf{Set})(p'[i'], p[i] +_{q[f(i)]} q'[j']) \\ &\cong \prod_{i \in p(1)} \prod_{\{j' \in q'(1) \mid g_1(j') = f_1(i)\}} \sum_{\{i' \in p'(1) \mid e_1(i') = i\}} (q[f(i)]/\mathbf{Set})(p'[i'], q'[j']) \\ &\cong \prod_{j \in q(1)} \prod_{\{j' \in q'(1) \mid g_1(j') = j\}} \prod_{\{i \in p(1) \mid f_1(i) = j\}} \sum_{\{i' \in p'(1) \mid e_1(i') = i\}} (q[j]/\mathbf{Set})(p'[i'], q'[j']) \\ &\cong \prod_{j \in q(1)} \prod_{\{j' \in q'(1) \mid g_1(j') = j\}} \sum_{i' \in \prod_{i \in f_1^{-1}(j)} e_1^{-1}(i)} \prod_{i \in f_1^{-1}(j)} (q[j]/\mathbf{Set})(p'[i'(i)], q'[j']) \\ &\cong \prod_{j \in q(1)} \prod_{\{j' \in q'(1) \mid g_1(j') = j\}} \sum_{i' \in \prod_{i \in f_1^{-1}(j)} e_1^{-1}(i)} (q[j]/\mathbf{Set}) \left( \sum_{i \in f_1^{-1}(j)} p'[i'(i)], q'[j'] \right) \\ &\cong (\mathbf{Poly}/q)(q', f_*p') \end{aligned}$$

□

*Example 4.69.* Let  $p := 2y^2$ ,  $q := y^2 + y^0$ , and  $f: p \rightarrow q$  the unique cartesian lens between them. Then for any  $e: p' \rightarrow p$  over  $p$ , (4.68) provides the following description for the pushforward  $f_*p'$ .

Over the  $j = 2$  position,  $f_1^{-1}(2) = 0$  and  $q[2] = 0$ , so  $\prod_{i \in f_1^{-1}(2)} e_1^{-1}(i)$  is an empty product and  $q[2]/\sum_{i \in f_1^{-1}(2)} p'[i'(i)]$  is an empty pushout. Hence the corresponding summand of (4.68) 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:

$$\begin{array}{ccc} X_{i'} & \xleftarrow{\quad} & p'[i'(2)] \\ \uparrow & \lrcorner & \uparrow e_{i'(2)}^\# \\ p'[i'(1)] & \xleftarrow[e_{i'(1)}^\#]{} & 2 \end{array}$$

Then in sum we have

$$f_*p' \cong \left( \sum_{i' \in e_1^{-1}(1) \times e_2^{-1}(2)} y^{X_{i'}} \right) + 1.$$

*Exercise 4.70.* Prove that the unique map  $f: y \rightarrow 1$  is exponentiable.  $\diamond$

For any set  $A$ , let  $A.\mathbf{Poly}$  denote the category whose objects are polynomials  $p$  equipped with an isomorphism  $A \cong p(1)$ , and whose morphisms are lenses respecting the isomorphisms with  $A$ .

**Proposition 4.71** (Base change). For any function  $f: A \rightarrow B$ , pullback  $f^*$  along  $f$  induces a functor  $B.\mathbf{Poly} \rightarrow A.\mathbf{Poly}$ , which we also denote  $f^*$ .

*Proof.* This follows from (4.38) with  $q := A$  and  $r := B$ , since pullback of an iso is an iso.  $\square$

**Theorem 4.72.** For any function  $f: A \rightarrow B$ , the pullback functor  $f^*$  has both a left and a right adjoint

$$\begin{array}{ccc} & \xrightarrow{f_!} & \\ & \Rightarrow & \\ A.\mathbf{Poly} & \xleftarrow{f^*} & B.\mathbf{Poly} \\ & \Leftarrow & \\ & \xrightarrow{f_*} & \end{array} \quad (4.73)$$

Moreover  $\otimes$  preserves the op-cartesian arrows, making this a monoidal \*-bifibration in the sense of [Shu08, Definition 12.1].

*Proof.* Let  $p$  be a polynomial with  $p(1) \cong A$ . Then the formula for  $f_!p$  and  $f_*p$  are given as follows:

$$f_!p \cong \sum_{b \in B} y^{a \mapsto b} \prod p[a] \quad \text{and} \quad f_*p \cong \sum_{b \in B} y^{a \mapsto b} \sum p[a] \quad (4.74)$$

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  $\mathbf{Poly}$  is equivalent to the Grothendieck construction applied to the functor  $\mathbf{Set}^{\text{op}} \rightarrow \mathbf{Cat}$  sending each set  $A$  to the category  $(\mathbf{Set}/A)^{\text{op}}$ . The fact that functions  $f: A \rightarrow B$  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 (4.74) and simultaneously establishes the adjoint triple (4.73).

The functor  $p \mapsto p(1)$  is strong monoidal with respect to  $\otimes$  and strict monoidal if we choose the lens construction as our model of **Poly**. By Proposition 4.62, the monoidal product  $\otimes$  preserves cartesian 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  $p \rightarrow f_!p$  obtained as the composite  $p \rightarrow f^*f_!p \rightarrow f_!p$  where the first map is the unit of the  $(f_!, f^*)$  adjunction and the second is the cartesian lens for  $f_!p$ . On positions  $p \rightarrow f_!p$  acts as  $f$ , and on directions it is given by projection.

If  $f: p(1) \rightarrow B$  and  $f': p'(1) \rightarrow B'$  are functions then we have

$$\begin{aligned} f_!(p) \otimes f'_!(p') &\cong \sum_{b \in B} \sum_{b' \in B'} y(\Pi_{a \mapsto b} p[a]) \times (\Pi_{a' \mapsto b'} p'[a']) \\ &\cong \sum_{(b, b') \in B \times B'} y(\Pi_{(a, a') \mapsto (b, b')} p[a] \times p'[b']) \\ &\cong (f_! \otimes f'_!)(p \otimes p') \end{aligned}$$

and the op-cartesian lenses are clearly preserved since projections in the second line match with projections in the first.  $\square$

## 4.7 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 4.14 is in fact distributive monoidal, and this comes up in work on entropy [Spi22] and on noncooperative strategic games [Cap22].

## **Part II**

# **A different category of categories**



---

# The composition product

---

We have seen that the category **Poly** of polynomial functors has quite a bit of well-interoperating mathematical structure. Further, it is an expressive way to talk about dynamical systems that can change their interfaces and wiring patterns based on their internal states.

But we touched upon one thing—what in some sense is the most interesting part of the story—only briefly. That thing is quite simple to state, and yet has profound consequences. Namely, polynomials can be composed:

$$y^2 \circ (y + 1) = (y + 1)^2 \cong y^2 + 2y + 1.$$

What could be simpler?

It turns out that this operation, which we'll see soon is a monoidal product, has a lot to do with time. There is a strong sense—made precise in Proposition 5.2—in which the polynomial  $p \circ q$  represents “starting at a position  $i$  in  $p$ , choosing a direction in  $p[i]$ , landing at a position  $j$  in  $q$ , choosing a direction in  $q[j]$ , and then landing... somewhere.” 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 3.43. We'll continue that story in Section 5.1.4.

The composition product has many surprises up its sleeve, as we'll see in the following chapters. We've given you a glimpse of some of them already in Section 1.5. We won't amass all the rest here; instead, we'll take you through the story step by step.

## 5.1 Defining the composition product

We begin with the definition of the composition product in terms of polynomials as functors.

### 5.1.1 Composite functors

**Definition 5.1** (Composition product). Given polynomial functors  $p, q$ , we let  $p \circ q$  denote their *composition product*, or their composite as functors. That is,  $p \circ q: \mathbf{Set} \rightarrow \mathbf{Set}$  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  $\mathbf{Set} \rightarrow \mathbf{Set}$ , but to check that the full subcategory **Poly** of  $\mathbf{Set}^{\mathbf{Set}}$  inherits this monoidal structure, we need to verify that the composite of two functors in **Poly** is still a functor in **Poly**.

**Proposition 5.2.** Suppose  $p, q \in \mathbf{Poly}$  are polynomial functors  $p, q: \mathbf{Set} \rightarrow \mathbf{Set}$ . Then their composite  $p \circ q$  is again a polynomial functor, and we have the following isomorphism:

$$p \circ q \cong \sum_{i \in p(1)} \prod_{a \in p[i]} \sum_{j \in q(1)} \prod_{b \in q[j]} y. \quad (5.3)$$

*Proof.* We can rewrite  $p$  and  $q$  as

$$p \cong \sum_{i \in p(1)} y^{p[i]} \cong \sum_{i \in p(1)} \prod_{a \in p[i]} y \quad \text{and} \quad q \cong \sum_{j \in q(1)} y^{q[j]} \cong \sum_{j \in q(1)} \prod_{b \in q[j]} y.$$

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 (5.3) is indeed the formula for the composite  $p \circ q$ . To see this is a polynomial, we use (2.27), which says we can rewrite the  $\prod \sum$  in (5.3) as a  $\sum \prod$  to obtain

$$p \circ q \cong \sum_{i \in p(1)} \sum_{\bar{j}: p[i] \rightarrow q(1)} y^{\sum_{a \in p[i]} q[\bar{j}(a)]} \quad (5.4)$$

(written slightly bigger for clarity), which is clearly a polynomial.  $\square$

**Corollary 5.5.** The category **Poly** has a monoidal structure  $(y, \circ)$ , where  $y$  is the identity functor and  $\circ$  is given by composition.

Because we may wish to use  $\circ$  to denote composition in arbitrary categories, we use a special symbol for polynomial composition, namely

$$p \triangleleft q := p \circ q.$$

The symbol  $\triangleleft$  looks a bit like the composition symbol in that it is an open shape, and when writing quickly by hand, it's okay if it morphs into a  $\circ$ . But  $\triangleleft$  highlights the asymmetry of composition, in contrast with the other monoidal structures on **Poly** we've encountered. Moreover, we'll soon see that  $\triangleleft$  is quite evocative in terms of trees.



For each  $n \in \mathbb{N}$ , we'll also use  $p^{\triangleleft n}$  to denote the  $n$ -fold composite of  $p$ , i.e.  $n$  copies of  $p$  all composed with each other.<sup>1</sup> In particular,  $p^{\triangleleft 0} = y$  and  $p^{\triangleleft 1} = p$ .

We repeat the important formulas from Proposition 5.2 and its proof in the new notation:

$$p \triangleleft q \cong \sum_{i \in p(1)} \prod_{a \in p[i]} \sum_{j \in q(1)} \prod_{b \in q[j]} y. \quad (5.6)$$

$$p \triangleleft q \cong \sum_{i \in p(1)} \sum_{\bar{j}: p[i] \rightarrow q(1)} y^{\sum_{a \in p[i]} q[\bar{j}(a)]} \quad (5.7)$$

*Exercise 5.8.* Let's consider (5.7) piece by piece, with concrete polynomials  $p := y^2 + y^1$  and  $q := y^3 + 1$ .

1. What is  $y^2 \triangleleft q$ ?
2. What is  $y^1 \triangleleft q$ ?
3. What is  $(y^2 + y^1) \triangleleft q$ ? This is what  $p \triangleleft q$  “should be.”
4. How many functions  $\bar{j}_1: p[1] \rightarrow q(1)$  are there?
5. For each function  $\bar{j}_1$  as above, what is  $\sum_{a \in p[1]} q[\bar{j}_1(a)]$ ?
6. How many functions  $\bar{j}_2: p[2] \rightarrow q(1)$  are there?
7. For each function  $\bar{j}_2$  as above, what is  $\sum_{a \in p[2]} q[\bar{j}_2(a)]$ ?
8. Write out

$$\sum_{i \in p(1)} \sum_{\bar{j}: p[i] \rightarrow q(1)} y^{\sum_{a \in p[i]} q[\bar{j}(a)]}.$$

Does the result agree with what  $p \triangleleft q$  should be?

◇

*Exercise 5.9.* 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 5.10.* Recall the closure operation  $[-, -]: \mathbf{Poly}^{\text{op}} \times \mathbf{Poly} \rightarrow \mathbf{Poly}$  for  $\otimes$  from (3.78). Show that for all  $A \in \mathbf{Set}$  and  $q \in \mathbf{Poly}$ , there is an isomorphism

$$y^A \triangleleft q \cong [Ay, q].$$

<sup>1</sup>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  $f: p \rightarrow p'$  and  $g: q \rightarrow q'$  between polynomial functors, their composition product  $f \triangleleft g: p \triangleleft q \rightarrow p' \triangleleft q'$  is given by *horizontal composition*.

**Definition 5.11** (Horizontal composition of natural transformations). Let  $f: p \rightarrow p'$  and  $g: q \rightarrow q'$  be two natural transformations between (polynomial) functors  $p, p', q, q': \mathbf{Set} \rightarrow \mathbf{Set}$ . Then the *horizontal composite* of  $f$  and  $g$ , denoted  $f \triangleleft g$ , is the natural transformation  $p \triangleleft q \rightarrow p' \triangleleft q'$  whose  $X$ -component for each  $X \in \mathbf{Set}$  is the function

$$p(q(X)) \xrightarrow{f_{q(X)}} p'(q(X)) \xrightarrow{p'(g_X)} p'(q'(X)) \quad (5.12)$$

obtained by composing the  $q(X)$ -component of  $f$  with the functor  $p'$  applied to the  $X$ -component of  $g$ .

*Exercise 5.13.* Show that we could have replaced the composite function (5.12) in Definition 5.11 with the function

$$p(q(X)) \xrightarrow{p(g_X)} p(q'(X)) \xrightarrow{f_{q'(X)}} p'(q'(X)) \quad (5.14)$$

obtained by composing  $p$  applied to the  $X$ -component of  $g$  with the  $q'(X)$ -component of  $f$ , without altering the definition. ◇

*Remark 5.15.* 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  $h \circ j: r \rightarrow t$  obtained by composing a lens  $h: r \rightarrow s$  with a lens  $j: s \rightarrow t$ , 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  $f \triangleleft g: p \triangleleft q \rightarrow p' \triangleleft q'$  obtained by applying the monoidal product functor  $\triangleleft: \mathbf{Poly} \times \mathbf{Poly} \rightarrow \mathbf{Poly}$  on the lenses  $f: p \rightarrow p'$  and  $g: q \rightarrow q'$ . 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.<sup>2</sup>

<sup>2</sup>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.

This is another reason we tend to avoid the symbol  $\circ$ , preferring to use  $\circ$  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 arenas or corolla forests.

### 5.1.2 Composite arenas

Let us interpret our formula (5.7) for the composite of two polynomials in terms of what it says about the positions and directions of the corresponding arenas. The position-set of  $p \triangleleft q$  is

$$(p \triangleleft q)(1) \cong \sum_{i \in p(1)} \sum_{\bar{j}: p[i] \rightarrow q(1)} 1 \cong \sum_{i \in p(1)} \mathbf{Set}(p[i], q(1)). \quad (5.16)$$

In other words, specifying a position of  $p \triangleleft q$  amounts to first specifying a  $p$ -position  $i$ , then specifying a function  $\bar{j}: p[i] \rightarrow q(1)$ , 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

$$(p \triangleleft q)[(i, \bar{j})] \cong \sum_{a \in p[i]} q[\bar{j}(a)]. \quad (5.17)$$

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  $p \triangleleft q$  as an arena, it may be a bit tricky to wrap your head around. Here is an alternative perspective that can help us get a better intuition for what's going on with composite arenas.

Back in Section 2.2.1, we saw how to write the instructions for choosing an element of a dependent sum or product of sets. For instance, given a polynomial  $p$  and a set  $X$ , the instructions for choosing an element of

$$p \triangleleft X = p(X) \cong \sum_{i \in p(1)} \prod_{a \in p[i]} X$$

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. Furthermore, to

highlight that these instructions are associated with some polynomial  $p$ , we will use our familiar arena terminology of positions and directions.

The instructions associated with a polynomial  $p$  are:

1. choose a  $p$ -position  $i$ ;
2. for each  $p[i]$ -direction  $a$ :
  - 2.1. choose a future.

If we think of polynomials in terms of their instructions, then (5.6) tells us that the composition product simply nests one set of instructions within another, as follows.

The instructions associated with a polynomial  $p \triangleleft q$  are:

1. choose a  $p$ -position  $i$ ;
2. for each  $p[i]$ -direction  $a$ :
  - 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 composite by nesting even further. We might think of such instructions as specifying some sort of length- $n$  *strategy*, in the sense of game theory, for picking positions given any directions—except that the opponent is somehow abstract, having no positions of its own.

When we rewrite (5.6) (5.7), we are collapsing the instructions down into the following, highlighting the positions and directions of  $p \triangleleft q$ .

The instructions associated with a polynomial  $p \triangleleft q$  are:

1. choose a  $p$ -position  $i$  and, for each  $p[i]$ -direction  $a$ , 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 5.1.3 that these instructions have a very natural interpretation when we translate from arenas to corolla forests.

*Exercise 5.18.* 1. Let  $p$  be an arbitrary polynomial. Write out the (uncollapsed) instructions associated with  $p^{\triangleleft 3} = p \triangleleft p \triangleleft p$ .

2. Write out the (uncollapsed) instructions for choosing an element of  $p \triangleleft p \triangleleft 1$ , but where you would normally write “choose an element of 1,” just write “done.”  $\diamond$

But how does the composition product act on lenses between arenas? Given lenses  $f: p \rightarrow p'$  and  $g: q \rightarrow q'$ , we can translate them to natural transformations, take their horizontal composite, then translate this back to a lens. The following exercise guides us through this process.

*Exercise 5.19* (The composition product of lenses). Fix lenses  $f: p \rightarrow p'$  and  $g: q \rightarrow q'$ . We seek to characterize their composition product  $f \triangleleft g: p \triangleleft q \rightarrow p' \triangleleft q'$ .

1. Use Proposition 2.71 to compute the  $q(X)$ -component of  $f$  as a natural transformation.
2. Use Propositions 2.47 and 2.71 to compute  $p'$  applied to the  $X$ -component of  $g$  as a natural transformation.
3. Combine #1 and #2 using Definition 5.11 to compute the horizontal composite  $f \triangleleft g$  of  $f$  and  $g$  as natural transformations.
4. Use Corollary 2.73 to translate the natural transformation  $f \triangleleft g$  obtained in #3 to a lens between arenas  $p \triangleleft q \rightarrow p' \triangleleft q'$ . Verify that for each  $(i, \bar{j}_i)$  in  $(p \triangleleft q)(1)$  (see (5.16)), its on-positions function sends

$$(i, \bar{j}_i) \xrightarrow{(f \triangleleft g)_1} (f_1(i), f_i^\# \circ \bar{j}_i \circ g_1); \quad (5.20)$$

while for each  $(a', b')$  in  $(p' \triangleleft q')[(f_1(i), f_i^\# \circ \bar{j}_i \circ g_1)]$  (see (5.17)), its on-directions function sends

$$(a', b') \xrightarrow{(f \triangleleft g)^\#_{(i, \bar{j}_i)}} (f_i^\#(a'), g_{\bar{j}_i(f_i^\#(a'))}^\#(b')). \quad (5.21)$$

◇

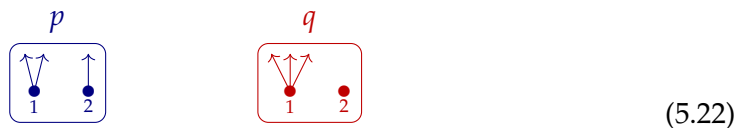
So what does Exercise 5.19 tell us about the behavior of  $f \triangleleft g: p \triangleleft q \rightarrow p' \triangleleft q'$ ? By (5.20), on positions,  $f \triangleleft g$  takes a  $p$ -position  $i$  and sends it to the  $p'$ -position  $f_1(i)$ ; then for each direction  $a'$  at this position, the associated  $q'$ -position is obtained by sending  $a'$  back to a  $p[i]$ -direction via  $f_i^\#$ , checking what  $q$ -position is associated to that  $p[i]$ -direction via some  $\bar{j}_i$ , then sending that  $q$ -position forward again to a  $q'$ -position via  $g_1$ .

Then by (5.20), 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 function of  $g$ . We'll get a better sense of what's happening when we see this drawn out as corolla forests in Example 5.31.

### 5.1.3 Composite corolla forests

It turns out that the forest of  $p \triangleleft q$  is given by gluing  $q$ -corollas onto the leaves of  $p$ -corollas in every possible way. We will demonstrate this using an example.

Let's say  $p := y^2 + y$  and  $q := y^3 + 1$ , whose corolla forests we draw as follows:



By (5.16), 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 gluing roots from the corolla

forest of  $q$  to leaves in the corolla forest of  $p$  in every possible way, as follows:

$$\begin{array}{c} \text{"}(p \triangleleft q)(1)\text{"} \\ \boxed{\begin{array}{cccccc} \begin{array}{c} \textcolor{red}{1} \quad \textcolor{red}{1} \\ \swarrow \quad \searrow \\ \textcolor{blue}{1} \end{array} & \begin{array}{c} \textcolor{red}{1} \quad \textcolor{red}{2} \\ \swarrow \quad \searrow \\ \textcolor{blue}{1} \end{array} & \begin{array}{c} \textcolor{red}{2} \quad \textcolor{red}{1} \\ \swarrow \quad \searrow \\ \textcolor{blue}{1} \end{array} & \begin{array}{c} \textcolor{red}{2} \quad \textcolor{red}{2} \\ \swarrow \quad \searrow \\ \textcolor{blue}{1} \end{array} & \begin{array}{c} \textcolor{red}{1} \\ \uparrow \\ \textcolor{blue}{2} \end{array} & \begin{array}{c} \textcolor{red}{2} \\ \uparrow \\ \textcolor{blue}{2} \end{array} \end{array}} \end{array} \quad (5.23)$$

Now fix one of the positions of  $p \triangleleft q$  drawn above: a  $p$ -root  $i$  and a  $q$ -root glued to every  $p[i]$ -leaf. By (5.17), 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 glued to  $a$ . In other words, in the following picture, where we have glued not just  $q$ -roots but entire  $q$ -corollas to leaves in  $p$ , the directions of  $p \triangleleft q$  at the position corresponding to each tree are the rooted paths<sup>3</sup> of that tree of length 2 (we omit the labels):

$$\begin{array}{c} \text{"}p \triangleleft q\text{"} \\ \boxed{\begin{array}{cccccc} \begin{array}{c} \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \\ \swarrow \quad \searrow \\ \textcolor{blue}{\bullet} \end{array} & \begin{array}{c} \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \\ \swarrow \quad \searrow \\ \textcolor{blue}{\bullet} \end{array} & \begin{array}{c} \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \\ \swarrow \quad \searrow \\ \textcolor{blue}{\bullet} \end{array} & \begin{array}{c} \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \\ \swarrow \quad \searrow \\ \textcolor{blue}{\bullet} \end{array} & \begin{array}{c} \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \\ \uparrow \\ \textcolor{blue}{\bullet} \end{array} & \begin{array}{c} \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \\ \uparrow \\ \textcolor{blue}{\bullet} \end{array} \end{array}} \end{array} \quad (5.24)$$

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 (5.24) because, to be pedantic, the corolla forest of  $p \triangleleft q$  has the two levels smashed together as follows:

$$\begin{array}{c} p \triangleleft q \\ \boxed{\begin{array}{cccccc} \begin{array}{c} \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \\ \bullet \end{array} & \begin{array}{c} \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \\ \bullet \end{array} & \begin{array}{c} \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \\ \bullet \end{array} & \bullet & \begin{array}{c} \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \quad \textcolor{red}{\nearrow} \\ \bullet \end{array} & \bullet \end{array}} \end{array} \quad (5.25)$$

Usually, we will prefer the style of (5.24) rather than the more pedantic style of (5.25).

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 5.1.2, where the arrows—the top-level leaves—are where the “futures” would go. Similarly, we could depict any  $n$ -fold composite as a forest with height- $n$  leaves as directions. You’ll have an opportunity to try this in the following exercise.

*Exercise 5.26.* Use  $p, q$  as in (5.22) and  $r := 2y + 1$  in the following.

1. Draw  $q \triangleleft p$ .
2. Draw  $p \triangleleft p$ .

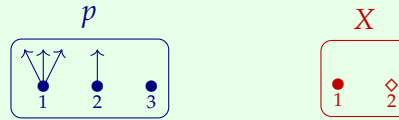
<sup>3</sup>A *rooted path* of a rooted tree is a path up the tree that starts from the root.

3. Draw  $p \triangleleft p \triangleleft 1$ .
4. Draw  $r \triangleleft r$ .
5. Draw  $r \triangleleft r \triangleleft r$ .

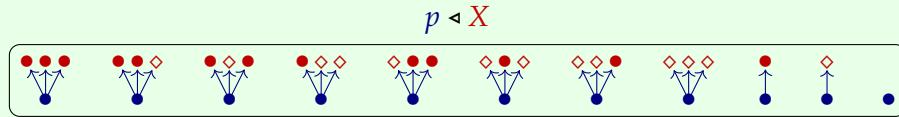
◇

*Example 5.27* (Composing polynomials with constants). For any set  $X$  and polynomial  $p$ , we can take  $p(X) \in \mathbf{Set}$ ; indeed  $p: \mathbf{Set} \rightarrow \mathbf{Set}$  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  $X := 2$ .

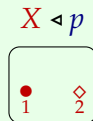


Let’s see how  $(y^3 + y + 1) \triangleleft 2$  looks.



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 5.28.*
1. Choose a polynomial  $p$  and draw  $p \triangleleft 1$  in the style of Example 5.27.
  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  $X$ .

◇

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 5.29.* Let  $\varphi: p \rightarrow q$  be a lens and  $X$  be a set viewed as a constant polynomial. Consider the lens  $\varphi \triangleleft X: p \triangleleft X \rightarrow q \triangleleft X$ , given by taking the composition product of  $\varphi$  with the identity lens on  $X$ . Show that  $\varphi \triangleleft X$ , when viewed as a function  $p(X) \rightarrow q(X)$  between sets, is exactly the  $X$ -component of  $\varphi$  viewed as a natural transformation.  $\diamond$

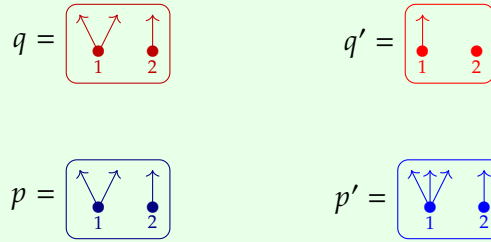
*Exercise 5.30.* For any  $p \in \mathbf{Poly}$  there are natural isomorphisms  $p \cong p \triangleleft y$  and  $p \cong y \triangleleft p$ .

1. Thinking of polynomials as functors  $\mathbf{Set} \rightarrow \mathbf{Set}$ , what functor does  $y$  represent?
2. Why are  $p \triangleleft y$  and  $y \triangleleft p$  isomorphic to  $p$ ?
3. Let  $p := y^3 + y + 1$ . In terms of tree pictures, draw  $p \triangleleft y$  and  $y \triangleleft p$ , and explain pictorially how to see the isomorphisms  $p \triangleleft y \cong p \cong y \triangleleft p$ .  $\diamond$

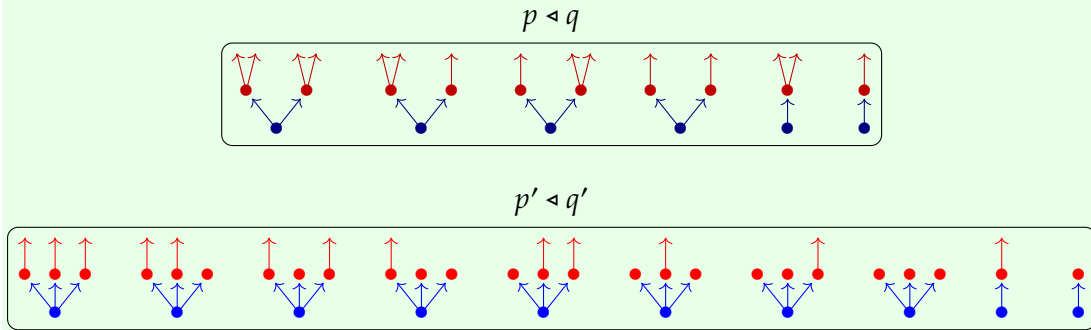
This is just  $p$  with every position propped up one level, so it is also still a picture of  $p$ .

How shall we think about taking the composition product of lenses in terms of our tree pictures? We can interpret the results of Exercise 5.19 as follows.

*Example 5.31.* Let's take  $p := y^2 + y$ ,  $q := y^2 + y$ ,  $p' := y^3 + y$ , and  $q' := y + 1$ .

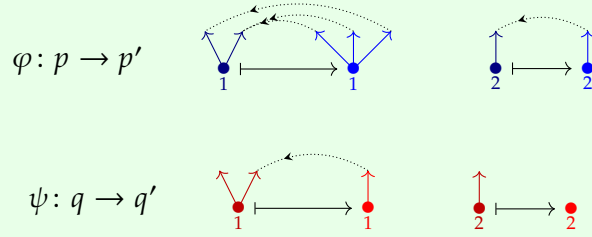


For any pair of lenses  $p \rightarrow p'$  and  $q \rightarrow q'$ , we have a lens  $p \triangleleft q \rightarrow p' \triangleleft q'$ . Let's draw  $p \triangleleft q$  and  $p' \triangleleft q'$ .





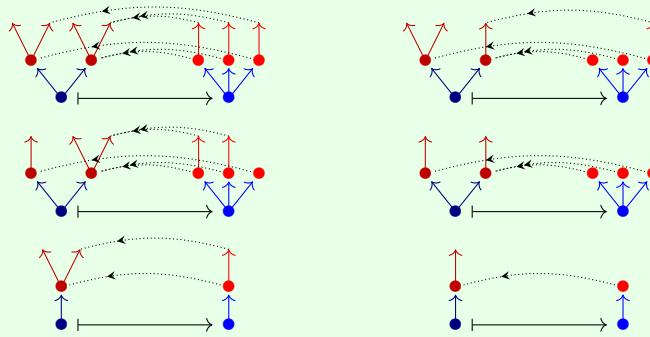
Let's also pick a pair of lenses,  $\varphi: p \rightarrow p'$  and  $\psi: q \rightarrow q'$ .



Then by Exercise 5.19, we can form the lens  $\varphi \triangleleft \psi: p \triangleleft q \rightarrow p' \triangleleft q'$  as follows. On positions, we follow (5.20): for each tree  $t$  in the picture of  $p \triangleleft q$ , we begin by using  $\varphi_1$  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 glued to  $a'$ , we use  $\varphi_i^\#$  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 glued to  $a$  in  $t$ , then use  $\psi_1$  to send  $j$  forward to a  $q'$ -corolla  $j'$ . This is the corolla we glue to 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 (5.21): 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  $\varphi_i^\#$  sends  $a'$  back to a height-1 vertex  $a$  of  $t$ , and as we saw, the on-positions function  $\psi_1$  sends the  $q$ -corolla  $j$  glued to  $a$  in  $t$  forward to the  $q'$ -corolla glued to  $a'$ . Then  $b'$  is a leaf of that  $q'$ -corolla, and  $\psi_j^\#$  sends  $b'$  back to a leaf  $b$  emanating from  $a$ . So the on-directions function  $(\varphi \triangleleft \psi)_t^\#$  sends the height-2 leaf  $b'$  to the height-2 leaf  $b$ .

We draw the lens  $\varphi \triangleleft \psi \rightarrow p \triangleleft q \rightarrow p' \triangleleft q'$  below. To avoid clutter, we leave out the arrows for  $\psi_1$  that show how the red corollas on the right are selected; we hope the reader can put it together for themselves.



*Exercise 5.32.* With  $p, q, p', q'$  and  $\varphi, \psi$  as in Example 5.31, draw the lens  $\psi \triangleleft \varphi: q \triangleleft p \rightarrow q' \triangleleft p'$  in terms of trees as in the example.  $\diamond$

*Exercise 5.33.* Suppose  $p$ ,  $q$ , and  $r$  are polynomials and you're given arbitrary lenses  $\varphi: q \rightarrow p \triangleleft q$  and  $\psi: q \rightarrow q \triangleleft r$ . Does the following diagram necessarily commute?<sup>4</sup>

$$\begin{array}{ccc} q & \xrightarrow{g} & q \triangleleft r \\ \varphi \downarrow & ? & \downarrow \varphi \triangleleft r \\ p \triangleleft q & \xrightarrow{p \triangleleft \psi} & p \triangleleft q \triangleleft r \end{array}$$

That is, do we have  $\varphi \circ (p \triangleleft \psi) = \psi \circ (\varphi \triangleleft r)$ ? ◇

### 5.1.4 Dynamical systems and the composition product

Back in Example 3.43, 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  $\varphi: Sy^S \rightarrow p$ , where  $S$  is a set of states and  $p$  is a polynomial interface. We call  $Sy^S$  the state system, each  $p$ -position  $i$  an output, each  $p[i]$ -direction an input at  $i$ , and the on-position and on-direction functions of  $\varphi$  the return and update functions, respectively. More generally, we saw in Example 3.43 that we could replace the state system with a monomial  $q := Sy^{S'}$ , where  $S'$  is another set, as long as there is a function  $e: S \rightarrow S'$  (or equivalently a situation  $\epsilon: Sy^{S'} \rightarrow y$ ) that is bijective.

The lens models a dynamical system as follows. Every state  $s \in q(1) = S$  returns an output  $o := \varphi_1(s) \in p(1)$ , and every input  $i \in p[o]$  yields an updated direction  $s' := \varphi_s^\#(a) \in q[s] = S'$ . 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  $\varphi_1$ , and repeat the process all over again.

But this is exactly what the composition product  $\varphi \triangleleft \varphi: q \triangleleft q \rightarrow p \triangleleft p$  does: by (5.20), 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  $e^{-1}: q[s_0] = S' \rightarrow S = q(1)$ , to the pair

$$\left( \varphi_1(s_0), \varphi_{s_0}^\# \circ e^{-1} \circ \varphi_1 \right) \in (p \triangleleft p)(1), \quad (5.34)$$

comprised of the initial output  $o_0 := \varphi_1(s) \in p(1)$  and a composite function

$$p[o_0] \xrightarrow{\varphi_{s_0}^\#} q[s_0] = S' \xrightarrow{e^{-1}} S = q(1) \xrightarrow{\varphi_1} p(1), \quad (5.35)$$

which uses the update function at  $s_0$  and the return function to tell us what the next output  $o_1$  will be for every possible input  $i_1$  we could select. Then by (5.21), the on-directions function of  $\varphi \triangleleft \varphi$  sends each direction  $(i_1, i_2)$  at the position (5.34), comprised

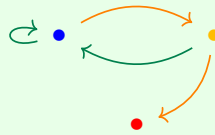
<sup>4</sup>When the name of an object is used in place of a morphism, we refer to the identity morphism on that object. So for instance,  $\varphi \triangleleft r$  is the composition product of  $\varphi$  with the identity lens on  $r$ .

of a first input  $i_1 \in p[o_0]$  and (setting  $o_1$  to be the function (5.35) applied to  $i_1$ ) a second input  $i_2 \in p[o_1]$ , to the pair

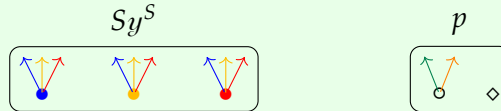
$$\left( \varphi_{s_0}^\#(i_1), \varphi_{e^{-1}(\varphi_{s_0}^\#(i_1))}^\#(i_2) \right),$$

comprised of directions in  $S'$  that (under  $e^{-1}$ ) correspond to the next state  $s_1$  upon selecting input  $i_1$  at state  $s_0$  and the successive state  $s_2$  upon selecting input  $i_2$  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 output  $o_0$ , receiving input  $i_1$ , updating its state to  $s_1$ , returning output  $o_1$ , receiving input  $i_2$ , and preparing to update its state to  $s_2$ . Adding another layer,  $\varphi \triangleleft \varphi \triangleleft \varphi: q \triangleleft q \triangleleft q \rightarrow p \triangleleft p \triangleleft p$  will tell us how the system behaves when we step through it three times; and in general,  $\varphi^{\triangleleft n}: q^{\triangleleft n} \rightarrow p^{\triangleleft n}$  will tell us how the system behaves when we step through it  $n$  times.

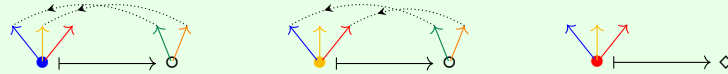
*Example 5.36* (Composition products of dynamical systems as trees). Consider the dynamical system  $\varphi: Sy^S \rightarrow p$  with  $p := y^A + 1$ , corresponding to the halting deterministic state automaton (3.22) from Exercise 3.21, depicted again here for convenience:



Below, we draw the corolla pictures for  $Sy^S$  and for  $p$ .

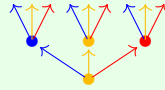


In the picture for  $Sy^S$ , the roots are the three states in  $\{\bullet, \bullet, \bullet\}$  appearing in the automaton, while the leaves of each corolla correspond to the three states as well. In the picture for  $p$ , there is one corolla whose two leaves correspond to the two arrows coming out of every state—except for the halting state, which is sent to the corolla with no leaves instead. So the lens  $\varphi: Sy^S \rightarrow p$  capturing the dynamics of the automaton can be drawn as follows:



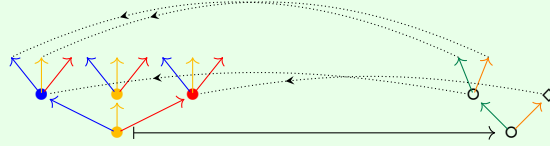
The corolla picture tells us, for example, that from the yellow state, we can go in one of two directions: the green direction, which leads us to the blue state, or the orange direction, which leads us to the red state. This describes the dynamics of the automaton one step away from the yellow state.

But what if we want to understand the dynamics of  $\varphi$  *two* steps away from the yellow state? Consider the following tree, corresponding to a position of the 2-fold composite  $Sy^S \triangleleft Sy^S$ :



(5.37)

We can follow the steps from Example 5.31 to find that the composition product of lenses  $\varphi \triangleleft \varphi: Sy^S \triangleleft Sy^S \rightarrow p \triangleleft p$  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 in the introduction when we said that the composition product has to do with time. It takes a specification  $\varphi$  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 input that the interface could encounter. Alternatively, we can think of  $\varphi^{\triangleleft n}$  as “speeding up” the original dynamical system  $\varphi$  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  $n$  inputs of  $p$  at a time! The lens  $\varphi$  tells us how the machine can run, but it is  $\triangleleft$  that makes the clock tick.

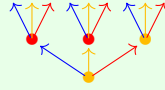
### Why this is not enough

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 3.43. 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 5.36), 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  $e^{-1}: S' \rightarrow S$  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  $f: S' \rightarrow S$  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 3.43, 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 5.38* (Composition products of dynamical systems can be misleading). In Example 5.38, instead of (5.37), we could have picked the following tree, corresponding to a different (yet entirely valid) position of  $Sy^S \triangleleft Sy^S$ :



The composition product  $\varphi \triangleleft \varphi: Sy^S \triangleleft Sy^S \rightarrow p \triangleleft p$  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 glued 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.

## 5.2 Lenses to composites

Lenses to composites—that is, lenses of the form  $f: p \rightarrow q_1 \triangleleft \cdots \triangleleft q_n$  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. Before we explore these properties, we'll introduce a new way of visualizing lenses that is well-suited to capturing the behavior of lenses to composites.

### 5.2.1 Lenses as polyboxes

First, let us consider what goes into specifying a lens  $f: p \rightarrow q$ . To visualize this, we will introduce a new form of notation, which we will call *polyboxes*, that will generalize well to lenses to composites:

$$\begin{array}{ccc}
 & f: p \rightarrow q & \\
 \begin{array}{c} p[-] \\ p(1) \\ p \end{array} & \begin{array}{c} \xleftarrow{\quad} \\ \xrightarrow{\quad} \end{array} & \begin{array}{c} q[-] \\ q(1) \\ q \end{array}
 \end{array} \tag{5.39}$$

In our polyboxes, each pair of boxes stacked on top of each other represents a single polynomial. The lower box in a pair can be filled with any position of the polynomial, while the upper box must be filled with a direction of the polynomial at the position in the lower box.

We think of (5.39) as depicting the lens  $f$  as a sort of gadget that acts like an automated spreadsheet: the blue boxes accept user input, while the white boxes are computed based on what is entered into the blue boxes according to the spreadsheet's preprogrammed rules. The arrows track the flow of information, starting from the lower left.

When the user fills the lower left blue box with a  $p$ -position  $i$ , the arrow to the right tells us that the gadget should automatically fill the lower right white box with some  $q$ -position  $j$ , based on the value  $i$  that has already been entered. This process yields a map  $i \mapsto j$  that corresponds to the on-position function  $f_1$  of the lens.

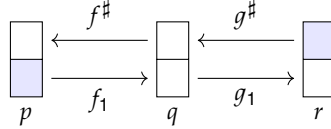
Then when the user fills the upper right blue box with a  $q[j]$ -direction  $b$ , the arrow to the left tells us that the gadget should automatically fill the upper left white box with some  $p[i]$ -position  $a$ , based on the values  $i$  and  $b$  that have already been entered. Fixing  $i \in p(1)$ , this process yields a map  $b \mapsto a$  that corresponds to the on-directions function  $f_i^\#$  of the lens.

So when both the user and the automation have finished filling all the boxes, we'll end up with something that looks like this:

$$\begin{array}{ccc}
 & f^\# & \\
 \begin{array}{c} a \\ i \end{array} & \begin{array}{c} \xleftarrow{\quad} \\ \xrightarrow{\quad} \end{array} & \begin{array}{c} b \\ j \end{array} \\
 p & & q
 \end{array}$$

Here, of course,  $j := f_1(i)$  and  $a := f_i^\#(b)$ . So a lens is any protocol that will fill in the white boxes once the user fills in the blue boxes, following the directions of the arrows drawn. Be careful: although the arrow  $f^\#$  is drawn from the upper right box, it also takes into account what is entered into the lower left box previously. After all, the on-directions function of a lens is dependent on both a position of the domain and a direction of the codomain.

If we have two composable lenses  $f: p \rightarrow q$  and  $g: q \rightarrow r$ , then we can piece their polyboxes together to form polyboxes for their composite,  $f \circ g: p \rightarrow r$ :



The lower (position) box for  $q$ , which would normally be blue as part of the polyboxes for  $g: q \rightarrow r$ , is instead filled in via  $f_1$ ; similarly, the upper (direction) box for  $q$ , which would normally be blue as part of the polyboxes for  $f: p \rightarrow q$ , is filled in via  $g^\#$ . This forms a gadget that is equivalent to what the polyboxes would be for  $f \circ g$ .

Again, as we follow the arrows from left to right and up and left again, take care to note that the arrow  $g^\#$  depends not only on the upper box for  $r$ , but also the lower box for  $q$  that came before it. Similarly, the arrow  $f^\#$  depends on both the lower box for  $p$  and the upper box for  $q$ .

On the other hand, the arrow  $g_1$  depends only on the lower box for  $q$ , and not the lower box for  $p$  that came before it:  $g_1$  is the on-positions function for a lens  $q \rightarrow r$  and therefore depends only on its domain. (Of course, changing the contents of the lower box for  $p$  may change the lower box for  $q$ , thus indirectly affecting what  $g_1$  enters in the lower box for  $r$ ; what we mean is that if the lower box for  $p$  changes but the lower box for  $q$  does not,  $g_1$  will not change the lower box for  $r$ .) For the same reason, the arrow  $g^\#$  does not depend on the lower box for  $p$ , and the arrow  $f^\#$  does not depend on either box for  $r$ . The key is to let each arrow depend on exactly the boxes that come before it in either the domain or the codomain of the lens that the arrow is a part of. Or just remember that these arrows are all on-positions and on-directions functions of lenses!

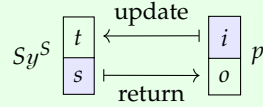
*Remark 5.40.* At this point in our introduction to polyboxes, the reader may be concerned that we are referring to things like “gadgets,” “spreadsheets,” “users,” “automation,” etc. without being entirely precise or rigorous about what we mean by them, or what it means to say that they are “equivalent.” We choose to elide this issue to highlight the pictorial intuition of our work, rather than grinding through the nitty-gritty details. This is not to say our work with polyboxes will lack rigor moving forward—if you’re particularly worried, you should think of polyboxes simply as an alternate way to present information about dependent sets, functions, sums, and products 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  $f: p \rightarrow q$  and  $g: q \rightarrow r$ , the polyboxes above really do just represent the element of the set

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

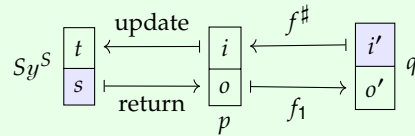
corresponding to the lens  $p \rightarrow r$  whose on-positions function  $p(1) \rightarrow r(1)$  is equal to the composite of the on-positions functions  $f_1$  and  $g_1$ , and whose on-directions function  $r[g_1(f_1(i))] \rightarrow p[i]$  for each  $i \in p(1)$  is equal to the composite of the on-directions functions  $g_{f_1(i)}^\#$  and  $f_i^\#$ . In other words, it represents the composite lens  $f \circ g$ . But it displays the way lenses pass positions and directions back and forth far more legibly than all the words in this paragraph can. Throughout the rest of this book, we'll 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 5.41* (Dynamical systems as polyboxes). Polyboxes provide a natural way to model our dependent dynamical systems. We can express such a system  $\varphi: Sy^S \rightarrow p$  in polyboxes as



Then the polybox acts 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 lower left blue box, and the return function converts this state to the output  $o \in p(1)$ , which is exposed by the interface in the lower right white box. Associated with this output is a set of inputs  $p[o]$ ; an interacting agent selects one of these inputs  $i \in p[o]$  to enter into the upper right blue box. Finally, the update function takes in the current state  $s$  and the input  $i$  and fills in the upper left white box accordingly with the next state  $t \in S$  (or, more precisely, a direction at the current state  $s$  that should point to the next state  $t$ ).

We then compose  $\varphi$  with a wrapper  $f: p \rightarrow q$  like so:

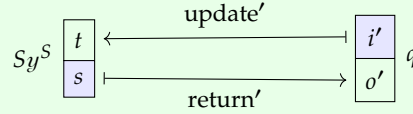


The output  $o$  displayed by the intermediary interface  $p$  is instead exposed as an output  $f_1(o) = o'$  of the wrapper interface  $q$  in the lower box on the far right. Moreover, the upper box of  $p$  is no longer blue: an agent who wishes to interact with the middle interface  $p$  can only do so via the rightmost interface  $q$ . The on-directions function



of the wrapper at  $o$  converts input  $i' \in q[o']$  from the upper right blue box into input  $i \in p[o]$ .

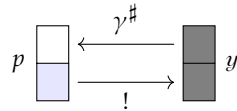
Picture the agent standing to the right of all the polyboxes (i.e. “outside” of the system) with their attention directed leftward (i.e. “inward”), receiving output from the white box below and feeding input into the blue box above. To an agent who is unaware of its inner workings, the composite dynamical system  $\varphi \circ f$  might as well look like this:



### 5.2.2 Situations as polyboxes

When a polynomial has only 1 position (i.e. it is representable), we shade its lower (position) polybox gray to indicate that there is no choice to be made there, either by the user or by the automation; similarly, if a polynomial has only 1 direction at every position (i.e. it is linear), we shade its upper (direction) polybox gray. So both polyboxes for  $y$  are shaded gray.

In Section 3.3.4, we discussed situations, which are lenses with codomain  $y$ . A situation  $\gamma: p \rightarrow y$ , then, can be depicted as follows, where  $!$  is the unique function into 1:



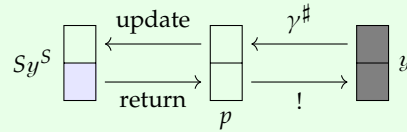
But this is equivalent to a gadget where, when the user fills in the lower left blue box with a  $p$ -position  $i$ , the upper left white box is automatically filled with a  $p[i]$ -direction  $a$ . (Remember that the arrow  $\gamma^\#$  depends not only on the box to its right, but also on the lower left box of  $p$ .) So we can redraw the gadget like so, combining the arrows  $!$  and  $\gamma^\#$  into one arrow  $\gamma$ :

$$p \begin{array}{|c|} \hline \phantom{a} \\ \hline \phantom{i} \\ \hline \end{array} \curvearrowright \gamma \quad (5.42)$$

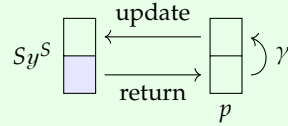
This lines up with what we already know: that a situation  $\gamma: p \rightarrow y$  is just a dependent function  $\gamma: (i \in p(1)) \rightarrow p[i]$ .

*Example 5.43* (Enclosures as polyboxes). Recall that in the language of dynamical systems, situations for interfaces are thought of as enclosures: a fixed selection of input at every output that closes off the interface from external observation or interference. In

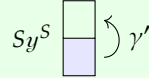
polyboxes, composing a system  $Sy^S \rightarrow p$  with an enclosure  $\gamma$  for  $p$  can be depicted as



or, equivalently, as



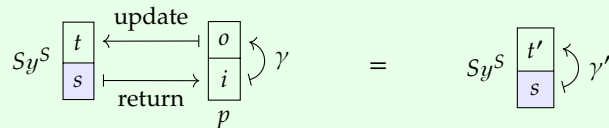
Remember: in a polybox depiction of a dynamical system, the world outside the system exists to the right of all the boxes. So the first picture represents  $y$  as a gray wall, cutting off any interaction between the system to its left and the world to its right. Meanwhile, the second picture illustrates how an enclosed system independently selects inputs to the intermediary interface  $p$  via  $\gamma$ , according to the outputs of  $p$  that the inner (leftward) system  $Sy^S \rightarrow p$  returns. While the second picture shows us why the closed system neither seeks nor requires external input, the first picture helps remind us that the output of  $p$  never reaches the outside world either. The composite system is therefore equivalent to the enclosure drawn as follows:



In polynomial morphism parlance,  $\gamma': Sy^S \rightarrow y$  is the original system  $Sy^S \rightarrow p$  composed with  $\gamma: p \rightarrow y$ ; in the language of dependent functions,  $\gamma': S \rightarrow S$  is given by

$$\gamma'(s) = \text{update}(s, \gamma(\text{return}(s))) \quad \text{for all } s \in S,$$

where we interpret  $\gamma$  as a dependent function  $(i \in p(1)) \rightarrow p[i]$ . We can deduce this equation by matching up the previous two different polybox pictures, knowing that they represent the same lens. Placing the boxes side by side and filling them in with dummy variables makes the equation easier to read off the picture:

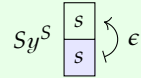


Notice that we filled the blue boxes on either side with the same entry. Matching up the upper boxes of the domain in both pictures, we have that  $t = t'$ , so

$$\gamma'(s) = t' = t = \text{update}(s, o) = \text{update}(s, \gamma(i)) = \text{update}(s, \gamma(\text{return}(s))).$$

Later on, we will read more intricate equations off of polyboxes in this manner, although we won't spell it out in so much detail. We encourage you to trace through the arrows on your own.

*Example 5.44* (The do-nothing enclosure in polyboxes). In Example 3.43, we saw that every state system  $Sy^S$  can be equipped with an enclosure  $\epsilon: Sy^S \rightarrow y$  called the do-nothing enclosure, which assigns each state-position to its corresponding state-direction, thus leaving the state unchanged. That is, it is the enclosure whose polyboxes can be drawn as follows:



This simple 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.

### 5.2.3 Lenses to composites as polyboxes

A lens  $p \rightarrow q_1 \triangleleft q_2$  is an element of the set

$$\mathbf{Poly}(p, q_1 \triangleleft q_2) \cong \mathbf{Poly}\left(p, \sum_{j_1 \in q_1(1)} \prod_{b_1 \in q_1[j_1]} \sum_{j_2 \in q_2(1)} \prod_{b_2 \in q_2[j_2]} y\right) \quad (5.6)$$

$$\cong \prod_{i \in p(1)} \sum_{j_1 \in q_1(1)} \prod_{b_1 \in q_1[j_1]} \sum_{j_2 \in q_2(1)} \prod_{b_2 \in q_2[j_2]} p[i]. \quad (2.55)$$

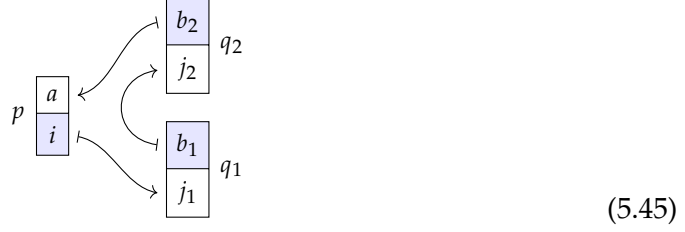
So we can write down the instructions for picking a lens  $p \rightarrow q_1 \triangleleft q_2$  as follows.

To choose a lens  $p \rightarrow q_1 \triangleleft q_2$ :

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:



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.

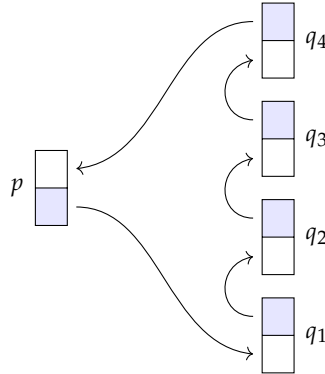
So a lens  $p \rightarrow q_1 \triangleleft q_2$  is any protocol that will fill in the white boxes above as the user fills in the blue boxes in the direction of the arrows. We’ll see this in action in Example 5.46.

In fact, (5.42), (5.39), and (5.45) are the respective polybox depictions of the  $n = 0, n = 1$ , and  $n = 2$  cases of lenses  $p \rightarrow q_1 \triangleleft \cdots \triangleleft q_n$  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

$$\mathbf{Poly}(p, q_1 \triangleleft \cdots \triangleleft q_n) \cong \mathbf{Poly}\left(p, \sum_{j_1 \in q_1(1)} \prod_{b_1 \in q_1[j_1]} \cdots \sum_{j_n \in q_n(1)} \prod_{b_n \in q_n[j_n]} y\right) \quad (5.6)$$

$$\cong \prod_{i \in p(1)} \sum_{j_1 \in q_1(1)} \prod_{b_1 \in q_1[j_1]} \cdots \sum_{j_n \in q_n(1)} \prod_{b_n \in q_n[j_n]} p[i], \quad (2.55)$$

so the polybox depiction of  $p \rightarrow q_1 \triangleleft \cdots \triangleleft q_n$  generalizes analogously. For example, here are the polyboxes corresponding to a lens to a 4-fold composite:

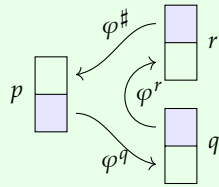


These lenses to  $n$ -fold composites lend themselves to a very natural interpretation in terms of our decision-making language. Each of  $p$ ’s menus is passed forward to a

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  $q_1, \dots, q_n$  chose then inform the option that  $p$  should select from its original menu.

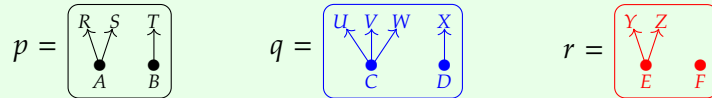
*A lens  $p \rightarrow q_1 \triangleleft \dots \triangleleft q_n$  is a multi-step policy for  $p$  to make decisions by asking for decisions from  $q_1$ , then  $q_2$ , etc., all the way to  $q_n$ , then interpreting the results.*

**Example 5.46** (Lenses  $p \rightarrow q \triangleleft r$ ). Consider a lens  $\varphi: p \rightarrow q \triangleleft r$ . Let's label the three arrows in the lens's polybox depiction:

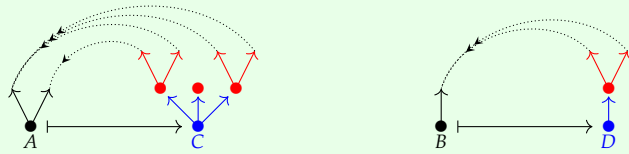


So the on-position function of  $\varphi$  can be split into two parts: a function  $\varphi^q: p(1) \rightarrow q(1)$  and, for each  $i \in p(1)$ , a function  $\varphi_i^r: q[\varphi^q(i)] \rightarrow r(1)$ . Then the on-directions function  $\varphi_i^\#: (q \triangleleft r)[\varphi_1(i)] \rightarrow p[i]$  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  $p := \{A\}y^{\{R,S\}} + By^{\{T\}}$ ,  $q := \{C\}y^{\{U,V,W\}} + \{D\}y^{\{X\}}$ , and  $r := \{E\}y^{\{Y,Z\}} + \{F\}$ .



Here is a tree picture of a lens  $\varphi: p \rightarrow q \triangleleft r$ :

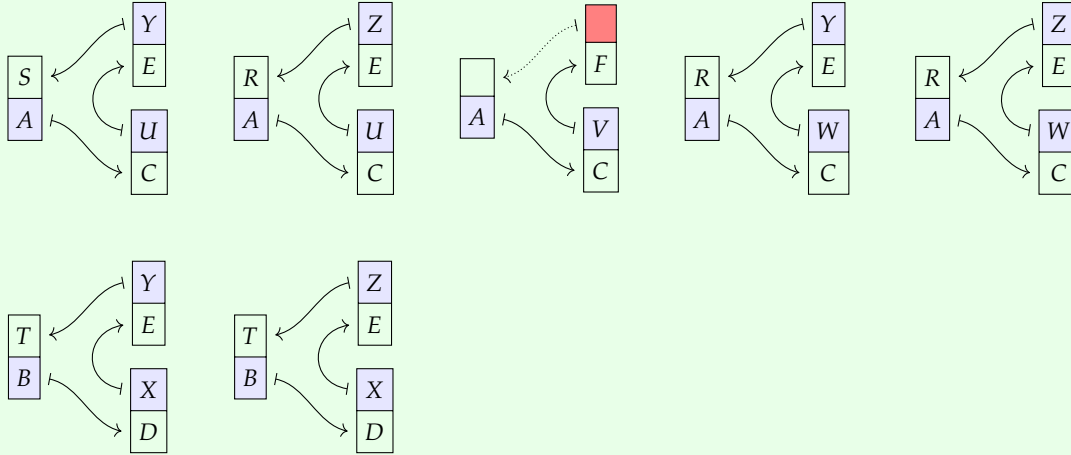


If we write  $\varphi$  as the corresponding triple  $(\varphi^q, \varphi^r, \varphi^\#)$ , then we have

$$\begin{aligned} \varphi^q(A) &= C, & \varphi^q(B) &= D; \\ \varphi_A^r(U) &= E, & \varphi_A^r(V) &= F, & \varphi_A^r(W) &= E; \\ \varphi_B^r(X) &= E; \end{aligned}$$

$$\begin{aligned}\varphi_A^\#(U, Y) = S, \quad \varphi_A^\#(U, Z) = R, \quad \varphi_A^\#(W, Y) = R, \quad \varphi_A^\#(W, Z) = R; \\ \varphi_B^\#(X, Y) = T, \quad \varphi_B^\#(X, Z) = T.\end{aligned}$$

Polyboxes display the same data in a different format:

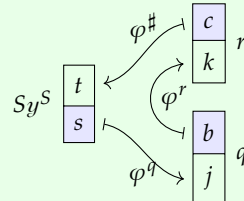


As before, keep in mind that each arrow of a lens depends not only on the box it emerges from, but also on every box that came before it in our usual reading order (lower left to lower right to upper right to upper left).

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  $\varphi_A^r(V) = F$ , but  $r[F] = \emptyset$ , 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 dashed.

*Example 5.47* (Dynamical systems with composite interfaces). We explored dynamical systems with product interfaces in Section 3.3.1 and parallel product interfaces in Section 3.3.2. 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  $\varphi: Sy^S \rightarrow q \triangleleft r$  can be drawn as



We can interpret the behavior of this system as follows. Rather than a single interface  $q \triangleleft r$ , we view  $\varphi$  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  $\varphi^q$  to return an output  $j$  of the first interface  $q$ . Upon receiving an input  $b \in q[j]$ , it then uses  $\varphi^r$  to return another output  $k$  (dependent on the state  $s$  and the input  $b$ ), this time belonging to the second interface  $r$ . Once a second input  $c$  is received, this time from  $r[k]$ , the system updates its state by feeding the current state  $s$  and the pair of inputs  $(b, c)$  it received into  $\varphi^\#$ , 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,  $\varphi$  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  $q_1 \triangleleft \cdots \triangleleft q_n$  produces output and receives input through interface  $q_1$ , then accordingly produces output and receives input through interface  $q_2$ , and so on, until it produces output (according to the current state and all previous inputs) and receives input through  $q_n$ , whereupon it updates its state according to the  $n$  inputs it received along with the current state.

#### 5.2.4 The composition product of lenses as polyboxes

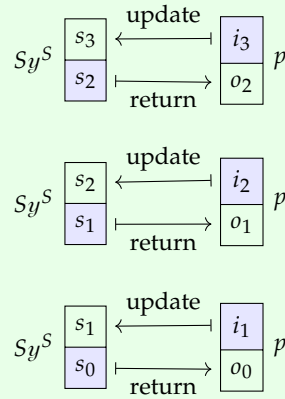
A special case of a lens whose codomain is a composite is a lens that is itself the composition product of lenses. If we draw such a lens using polyboxes by following the instructions from (5.20) and (5.21), we would really just be stacking the polyboxes for the constituent lenses on top of each other. For example, given lenses  $\varphi: p \rightarrow q$  and  $\varphi': p' \rightarrow q'$ , here is  $\varphi \triangleleft \varphi'$  drawn as polyboxes:

$$\begin{array}{ccc}
 \begin{array}{|c|} \hline \square \\ \hline \square \\ \hline \end{array} & \begin{array}{c} \xleftarrow{(\varphi')^\#} \\ \xrightarrow{\varphi'_1} \end{array} & \begin{array}{|c|} \hline \square \\ \hline \square \\ \hline \end{array} \\
 p' & & q' \\
 \\ 
 \begin{array}{|c|} \hline \square \\ \hline \square \\ \hline \end{array} & \begin{array}{c} \xleftarrow{\varphi^\#} \\ \xrightarrow{\varphi_1} \end{array} & \begin{array}{|c|} \hline \square \\ \hline \square \\ \hline \end{array} \\
 p & & q
 \end{array} \tag{5.48}$$

What differentiates this from simply writing down the polyboxes for  $\varphi$  and the polyboxes for  $\varphi'$  is that we are explicitly 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  $\varphi_1$  and  $\varphi^\#$  to decide what to put in the lower box of  $p'$ . So this really does depict a lens  $p \triangleleft p' \rightarrow q \triangleleft q'$ .

How does (5.48) relate to our usual polybox depiction of a lens to a composite, as in (5.45), but with the domain also replaced with a composite? A user who interacts with (5.48) can fill the lower set of polyboxes (the ones for  $\varphi$ ) first, ignoring the upper set of polyboxes (the ones for  $\varphi'$ ) 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 (5.16), such a choice is equivalent to picking a position of the composite  $p \triangleleft p'$ . Then by (5.20), following just the bottom arrow  $\varphi_1$  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  $\varphi'_1$  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, (5.48) can be thought of as a special case of (5.45).

*Example 5.49* (Dynamical systems and the composition product, revisited). In Section 5.1.4, we explained how the  $n$ -fold composition product  $\varphi^{\triangleleft n}$  of a dynamical system  $\varphi: Sy^S \rightarrow p$  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:

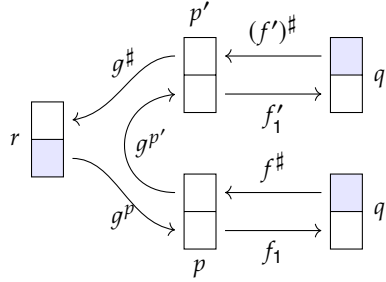


It is now patently obvious what  $\varphi^{\triangleleft 3}$  does from this picture, as long as we know how to read polyboxes (and we could probably make a pretty good guess even if we didn't!). This resolves the first issue we raised in Section 5.1.4, page 128: we now have a concise way of depicting 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 5.50.

We make a big deal out of it, but (5.48) really is just the polyboxes of two separate

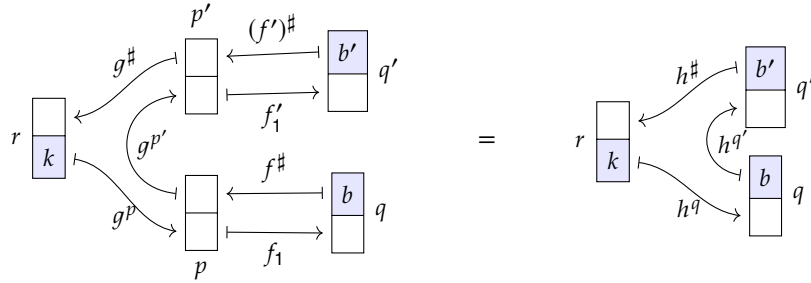


lenses drawn together. Where such polyboxes truly get interesting is when we compose them with polyboxes that look like (5.45). That is, given a lens  $g: r \rightarrow p \triangleleft p'$ , consider the polyboxes for  $g \circ (f \triangleleft f')$ :



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  $g \circ (f \triangleleft f')$  were equal to some other lens  $h: r \rightarrow q \triangleleft q'$ :



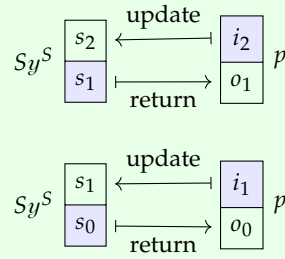
We've filled in the corresponding blue boxes on either side of the equation with the same entries. So if these two sets of polyboxes really do depict the same lens, each of the three white boxes in the domain and codomain on the left should end up with the same entry as the corresponding white box on 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:

$$g^p \circ f_1 = h^q, \quad f_{g^p(k)}^\# \circ g_k^{p'} \circ f_1' = h_k^{q'}, \quad \text{and} \quad (f')^\#_{g^{p'}(f_{g^p(k)}^\#(b))} \circ g_k^\# = h_k^\#.$$

The converse holds as well: if the three equations above all hold, then  $g \circ (f \triangleleft f') = h$ . We will read equations off of polyboxes like this repeatedly in the rest of the book.

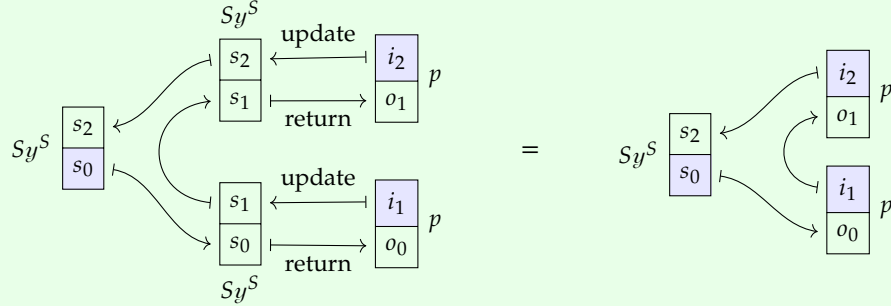
*Example 5.50* (Transition lenses for state systems). In Example 5.49, we saw that the 2-fold composition product  $\varphi^{\triangleleft 2}$  of a dynamical system  $\varphi: Sy^S \rightarrow p$  can be drawn as

follows:

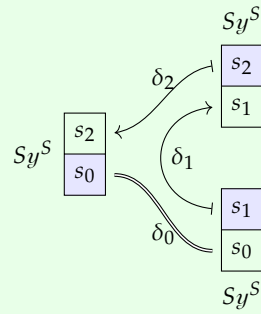


This *almost* models the behavior of running through the system twice, except that we should really only have one blue box on the domain side—the one we fill with the initial state  $s_0$ . The second blue box on the domain side, the one we fill with  $s_1$ , should instead be filled automatically with 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  $Sy^S \rightarrow p \triangleleft p$  that takes an initial state  $s_0 \in S$  and runs the original system  $\varphi$  twice, returning two outputs and receiving two inputs before stopping at the new state  $s_2$ . But we just learned how to take a composition product of lenses such as  $\varphi^{\triangleleft 2}: Sy^S \triangleleft Sy^S \rightarrow p \triangleleft p$  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  $Sy^S \rightarrow Sy^S \triangleleft Sy^S$  like so:



We'll denote our new lens  $Sy^S \rightarrow Sy^S \triangleleft Sy^S$  on the far left by  $\delta$ . Let's take a closer look at how  $\delta$  behaves on its own, labeling its arrows for ease of reference:



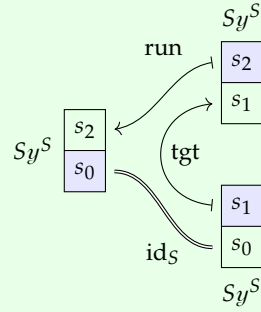
We can see from this picture that  $\delta$  arises very naturally from the structure of  $Sy^S$ ; indeed, every state system can be equipped with such a lens, just as every state system can be equipped with a do-nothing enclosure from Example 3.43. The bottom arrow  $\delta_0: S \rightarrow S$  sending  $s_0 \mapsto s_0$  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  $\delta_1$  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  $\delta_1: S \times S \rightarrow S$  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,  $\delta_2: S \times S \times S \rightarrow S$  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  $\delta_1$  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  $\delta_1$  that tells us the new state  $\delta_1(s, t) = t$  that  $(s, t)$  transitions to. For that reason, we call  $\delta$  the *transition lens* of the state system  $Sy^S$ , and we call  $\delta_1: S \times S \rightarrow S$  the *target function*, relabeling it *tgt* for short, indicating where each direction leads.

This is exactly what we wanted back in Example 3.43: 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  $\delta$ , and polyboxes like the ones above are the way we write them down.

We highlighted  $\delta_1$  above, but the arrows  $\delta_0$  and  $\delta_2$  are no less important. As an identity function,  $\delta_0 = \text{id}_S$  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,  $\delta_2$  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  $\delta$  of  $Sy^S$  with our new arrow names:



This resolves the second issue we raised in Section 5.1.4, page 128 for the case of  $n = 2$ , giving us a dynamical system  $\delta \circ (\varphi \triangleleft \varphi): Sy^S \rightarrow p \triangleleft p$  that simulates stepping through the system  $\varphi$  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 enclosure  $\epsilon: Sy^S \rightarrow y$  models. But there is some overlap in how  $\epsilon$  matches up state-positions with directions and how  $\delta$  does. Put another way, if  $\epsilon$  tells us how to do nothing, and  $\delta$  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  $\epsilon$  and  $\delta$  agree on what they are saying about our state system?

Then for  $n = 3$ , we would like a dynamical system  $Sy^S \rightarrow p \triangleleft p \triangleleft p$  that simulates stepping through  $\varphi$  three times. One way to do this would be to compose  $\varphi^{\triangleleft 3}$  with a lens of the form  $Sy^S \rightarrow (Sy^S)^{\triangleleft 3}$  that we obtain by extending  $\delta$  from modeling two-step transitions to three-step transitions. But there are two ways to derive a lens with codomain  $(Sy^S)^{\triangleleft 3}$  from  $\delta$ : we could either take  $\delta \circ (\delta \triangleleft \text{id}_{Sy^S})$ , or we could take  $\delta \circ (\text{id}_{Sy^S} \triangleleft \delta)$ . 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  $\varphi$  a fixed number of times. How do we guarantee that all these different ways of extending  $\delta$  to  $n$ -step transitions end up telling us the same thing?

In summary, we need some kind of compatibility condition between  $\epsilon$  and  $\delta$ , as well as some kind of associativity condition on  $\delta$  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 5.51.* Let  $S := \mathbb{N}$  and  $p := \mathbb{R}y^1$ , and define  $\varphi: Sy^S \rightarrow p$  to be the dynamical system with return function  $\varphi_1(k) := k$  and update function  $\varphi_k^\#(1) := k + 1$ .

1. Draw the polyboxes for  $\varphi$  and describe its dynamics: what does 1 run through the system look like?
2. Let  $\delta: Sy^S \rightarrow Sy^S \triangleleft Sy^S$  be the transition lens of  $Sy^S$ , and draw the polyboxes

for the new system  $\delta \circ (\varphi \triangleleft \varphi): Sy^S \rightarrow p \triangleleft p$ . Describe its dynamics: how does it model 2 runs through the system?  $\diamond$

*Exercise 5.52.* As a lens whose domain is a state system, the transition lens  $\delta: Sy^S \rightarrow Sy^S \triangleleft Sy^S$  of a state system  $Sy^S$  can be interpreted as a standalone dynamical system. Describe the dynamics of this system.  $\diamond$

## 5.3 Categorical properties of the composition product

We conclude this chapter by discussing several interesting properties of the composition product, many of which will come in handy in the following chapters. We'll focus on how  $\triangleleft$  interacts with other constructions on **Poly** that we introduced in previous chapters.

### 5.3.1 Interaction with products and coproducts

It turns out that the composition product behaves well—albeit asymmetrically—with products and coproducts.

**Proposition 5.53** (Left distributivity of  $\triangleleft$  over  $+$  and  $\times$ ). Given a polynomial  $r$ , the functor  $(-\triangleleft r): \mathbf{Poly} \rightarrow \mathbf{Poly}$  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:

$$(p + q) \triangleleft r \cong (p \triangleleft r) + (q \triangleleft r) \quad (5.54)$$

and

$$pq \triangleleft r \cong (p \triangleleft r)(q \triangleleft r). \quad (5.55)$$

More generally, given a set  $A$  and polynomials  $(q_a)_{a \in A}$ , we have the following natural isomorphisms:

$$\left( \sum_{a \in A} q_a \right) \triangleleft r \cong \sum_{a \in A} (q_a \triangleleft r) \quad (5.56)$$

and

$$\left( \prod_{a \in A} q_a \right) \triangleleft r \cong \prod_{a \in A} (q_a \triangleleft r) \quad (5.57)$$

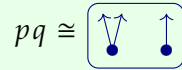
*Proof.* Formally, this comes down to the fact that (co)products of functors  $\mathbf{Set} \rightarrow \mathbf{Set}$  are computed pointwise (Proposition 2.34) and that (co)products in **Poly** coincide with (co)products in  $\mathbf{Set}^{\mathbf{Set}}$  (Propositions 2.51 and 2.80). One could instead give an explicit proof using (5.6); this is done in Exercise 5.58. In fact, we will see yet another proof of (5.57) (and thus (5.55)) in Exercise 5.76 #2.  $\square$

*Exercise 5.58.* Prove Proposition 5.53 using the explicit formula for  $\triangleleft$  given in (5.6) by manipulating sums and products.  $\diamond$

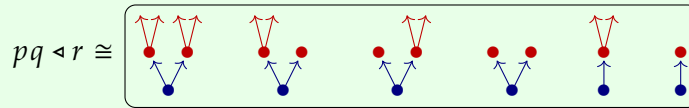
*Example 5.59* (Picturing the left distributivity of  $\triangleleft$  over  $\times$ ). We want an intuitive understanding of the left distributivity given by (5.55). Let  $p := y$ ,  $q := y + 1$ , and  $r := y^2 + 1$ , as shown here:



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:

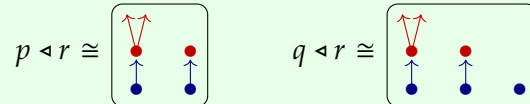


We can therefore draw  $pq \triangleleft r$  by gluing  $r$ -corollas to leaves of  $pq$  in every way, as follows:

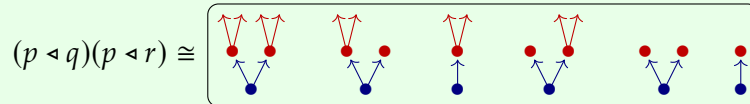


So each tree in  $pq \triangleleft r$  is obtained by gluing together the roots of a  $p$ -corolla and a  $q$ -corolla, then attaching  $r$ -corollas to each leaf.

Alternatively, we can compute  $p \triangleleft r$  and  $q \triangleleft r$  separately, gluing  $r$ -corollas to leaves of  $p$  in every way, then to leaves of  $q$  in every way:



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:



So each tree in  $(p \triangleleft r)(q \triangleleft r)$  is obtained by attaching  $r$ -corollas to each leaf of a  $p$ -corolla and a  $q$ -corolla before gluing their roots together.

But it doesn't matter if we glue  $r$ -corollas to 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 5.60.* Follow Example 5.59 with coproducts (+) in place of products ( $\times$ ): use pictures to give an intuitive understanding of the left distributivity given by (5.54).  $\diamond$

*Exercise 5.61.* 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 5.3.2, we will see how to generalize the left distributivity of  $\triangleleft$  over products to arbitrary limits. But first, we observe that right distributivity does not hold.

*Exercise 5.62.* Show that the distributivities of Proposition 5.53 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.

### 5.3.2 Interaction with limits on the left

We saw in Theorem 4.32 that **Poly** has all limits, and we saw in Exercise 4.41 that these limits coincide with limits in **Set**<sup>Set</sup>. Hence the argument in the proof of Proposition 5.53 by appealing to Proposition 2.34 can be generalized to arbitrary limits. It follows that  $\triangleleft$  preserves all limits on the left. But we will present a proof of this fact from an alternative perspective: by appealing to the left coclosure of  $\triangleleft$ .

**Proposition 5.63** (Meyers). The composition product is left coclosed. That is, there exists a left coclosure operation, which we denote  $\left[ \_ \right] : \mathbf{Poly}^{\text{op}} \times \mathbf{Poly} \rightarrow \mathbf{Poly}$ , such that there is a natural isomorphism

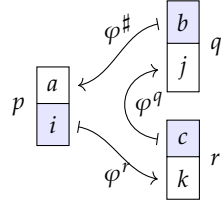
$$\mathbf{Poly}(p, r \triangleleft q) \cong \mathbf{Poly} \left( \left[ \begin{array}{c} q \\ p \end{array} \right], r \right). \quad (5.64)$$

In particular, the left coclosure operation sends  $q, p \in \mathbf{Poly}$  to

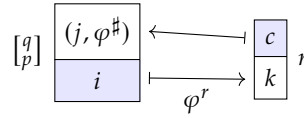
$$\left[ \begin{array}{c} q \\ p \end{array} \right] := \sum_{i \in p(1)} y^{q(p[i])}. \quad (5.65)$$

*Proof.* We present an argument using polyboxes; we leave it to the reader to write this proof in more standard mathematical notation in Exercise 5.66.

As in Example 5.46, a lens  $\varphi: p \rightarrow r \triangleleft q$  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  $\varphi^r$ , 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  $\varphi^q$  and  $\varphi^\#$  into a single on-directions arrow and directions box):



Here the on-directions function encodes the behaviors of both  $\varphi^q$  and  $\varphi^\#$  by sending each  $r[k]$ -direction  $c$  to both a  $q$ -position  $j$ , as  $\varphi^q$  does, and a function sending  $q[j]$ -directions to  $p[i]$ -directions, as  $\varphi^\#$  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  $(j, \varphi^\#)$  consisting of a  $q$ -position  $j$  and a function  $\varphi^\#: q[j] \rightarrow p[i]$ . 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  $\begin{bmatrix} q \\ p \end{bmatrix}$  in (5.65). It follows that there is a natural isomorphism between lenses  $p \rightarrow r \triangleleft q$  and lenses  $\begin{bmatrix} q \\ p \end{bmatrix} \rightarrow r$ .  $\square$

*Exercise 5.66.* Translate the polyboxes proof of Proposition 5.63 into standard mathematical notation, i.e. the  $\sum$  and  $\prod$  notation we have been using up till now.  $\diamond$

*Remark 5.67.* The proof you came up with in Exercise 5.66 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  $p \rightarrow r \triangleleft q$  can be simply repackaged and transferred to the domain-side of a new lens  $\begin{bmatrix} q \\ p \end{bmatrix} \rightarrow r$ .

*Exercise 5.68.* In stating Proposition 5.63, we implicitly assumed that  $\begin{bmatrix} - \\ - \end{bmatrix}$  is a functor  $\mathbf{Poly}^{\text{op}} \times \mathbf{Poly} \rightarrow \mathbf{Poly}$ . Here we show that this is indeed the case.

1. Given a polynomial  $q$  and a lens  $\varphi: p \rightarrow p'$ , to what lens  $\begin{bmatrix} q \\ p \end{bmatrix} \rightarrow \begin{bmatrix} q \\ p' \end{bmatrix}$  should the covariant functor  $\begin{bmatrix} q \\ - \end{bmatrix}$  send  $\varphi$ ? Prove that your construction is functorial.
2. Given a polynomial  $p$  and a lens  $\psi: q' \rightarrow q$ , to what lens  $\begin{bmatrix} q \\ p \end{bmatrix} \rightarrow \begin{bmatrix} q' \\ p \end{bmatrix}$  should the



contravariant functor  $[-]_p$  send  $\psi$ ? Prove that your construction is functorial.  $\diamond$

*Exercise 5.69.* In personal communication, Todd Trimble noted that the left coclosure can be thought of as a left Kan extension

$$\begin{array}{ccc} \mathbf{Set} & \xrightarrow{p} & \mathbf{Set} \\ q \downarrow & \Downarrow & \nearrow [q]_p \\ \mathbf{Set} & & \end{array}$$

Verify this.  $\diamond$

*Exercise 5.70.* Let  $A$  and  $B$  be sets, and let  $p$  and  $q$  be polynomials.

1. Prove that the following natural isomorphism holds:

$$\mathbf{Poly}(Ay^B, p) \cong \mathbf{Set}(A, p(B)). \quad (5.71)$$

2. Prove that the following natural isomorphism holds:

$$\mathbf{Poly}(Ay \triangleleft p \triangleleft y^B, q) \cong \mathbf{Poly}(p, y^A \triangleleft q \triangleleft By). \quad (5.72)$$

(Hint: Break the isomorphism down into two parts. You may find (4.9) helpful.)  $\diamond$

*Example 5.73* (Dynamical systems as coalgebras). Taking  $A = B = S \in \mathbf{Set}$  in (5.71), we find that there is a natural isomorphism between dynamical systems  $Sy^S \rightarrow p$  and functions  $S \rightarrow p(S)$ . Such a function is known as a *coalgebra for the functor  $p$*  or a  *$p$ -coalgebra*.<sup>5</sup>

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  $\mathbf{Set} \rightarrow \mathbf{Set}$  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  $Sy^S \rightarrow p$  rather than as functions  $S \rightarrow p(S)$  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: **Poly** lets us talk about both.

**Proposition 5.74** (Left preservation of limits). The operation  $\triangleleft$  preserves limits on the left (up to natural isomorphism). That is, if  $\mathcal{G}$  is a category,  $p_- : \mathcal{G} \rightarrow \mathbf{Poly}$  is a functor, and  $q \in \mathbf{Poly}$  is a polynomial, then there is a natural isomorphism

$$\left( \lim_{j \in \mathcal{G}} p_j \right) \triangleleft q \cong \lim_{j \in \mathcal{G}} (p_j \triangleleft q). \quad (5.75)$$

*Proof.* By Proposition 5.63, the functor  $(- \triangleleft q) : \mathbf{Poly} \rightarrow \mathbf{Poly}$  is the right adjoint of the functor  $[q] : \mathbf{Poly} \rightarrow \mathbf{Poly}$ , and right adjoints preserve limits.  $\square$

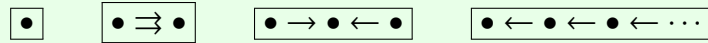
*Exercise 5.76.* 1. Complete Exercise 5.28 #3 using (5.75) and (5.56).  
2. Deduce (5.57) using (5.75).  $\diamond$

### 5.3.3 Interaction with limits on the right

So  $\triangleleft$  preserves limits on the left. How about limits on the right? We saw in Exercise 5.62 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 5.77** (Connected limit). A *connected limit* is one whose indexing category  $\mathcal{G}$  is nonempty and connected. That is,  $\mathcal{G}$  has at least one object, and any two objects are connected by a finite zigzag of arrows.

*Example 5.78.* The following categories are connected:



In particular, equalizers, pullbacks, and directed limits are examples of connected limits.

The following categories are *not* connected:



In particular, terminal objects and products are *not* examples of connected limits.

Connected limits are intimately related to slice categories, which we defined back in Definition 4.65. For example, products in a slice category  $\mathcal{C}/c$  are just pullbacks in

<sup>5</sup>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 6.3.3. 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.

$\mathcal{C}$ , 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 (5.80) below,  $p$  represents the lens  $f: p \rightarrow q \triangleleft 1$  and  $q \triangleleft r$  represents the lens  $q \triangleleft !: q \triangleleft r \rightarrow q \triangleleft 1$ , both objects in the slice category **Poly**/ $q \triangleleft 1$ .

**Proposition 5.79.** Given polynomials  $p, q, r \in \mathbf{Poly}$  and a function  $f: p \rightarrow q \triangleleft 1$ , there is a natural isomorphism

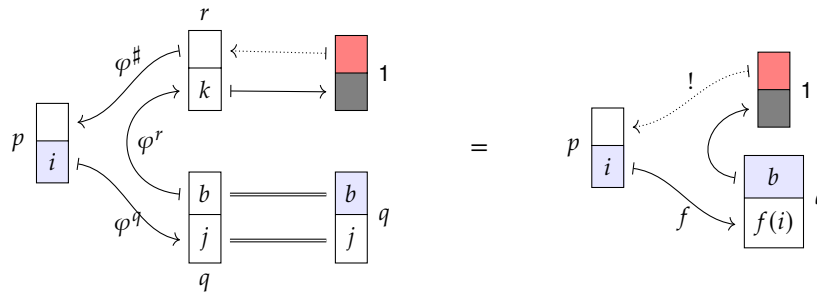
$$\mathbf{Poly}/(q \triangleleft 1)(p, q \triangleleft r) \cong \mathbf{Poly}\left(p \overset{f}{\frown} q, r\right), \quad (5.80)$$

where

$$p \overset{f}{\frown} q := \sum_{i \in p(1)} q[f(i)]y^{p[i]}. \quad (5.81)$$

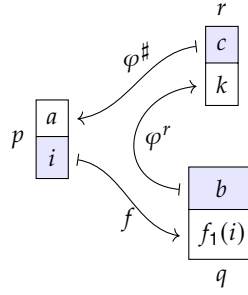
*Proof.* Again, we present an argument using polyboxes; we leave it to the reader to write this proof in more standard mathematical notation in Exercise 5.83.

Note that to consider  $q \triangleleft r$  as an object in **Poly**/ $(q \triangleleft 1)$ , we are implicitly using the map  $q \triangleleft !: q \triangleleft r \rightarrow q \triangleleft 1$ . By definition, morphisms from  $f$  to  $q \triangleleft !$  in **Poly**/ $(q \triangleleft 1)$  are lenses  $\varphi: p \rightarrow q \triangleleft r$  for which  $\varphi \circ (q \triangleleft !) = f$ . We can write this equation using polyboxes:

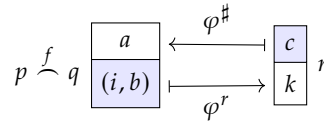


We can read off the picture that a lens  $\varphi: p \rightarrow q \triangleleft r$  is a morphism from  $f$  to  $q \triangleleft !$  in

**Poly**/ $q \triangleleft 1$  if and only if  $\varphi^q = f_1$ . So morphisms from  $f$  to  $q \triangleleft 1$  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  $\varphi^\#$ , 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  $\varphi^r$  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  $\varphi^r$ . 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 (5.81), so the isomorphism holds.  $\square$

*Remark 5.82.* As a lens  $p \rightarrow q \triangleleft 1$  can be identified with its on-positions function  $p(1) \rightarrow q(1)$ , we'll use the notation  $p \overset{f}{\curvearrowright} q$  interchangeably for lenses  $f: p \rightarrow q \triangleleft 1$  and functions  $f: p(1) \rightarrow q(1)$ .

*Exercise 5.83.* 1. Translate the polyboxes proof of Proposition 5.79 into standard mathematical notation.

2. Prove that the following natural isomorphism holds:

$$\mathbf{Poly}(p, q \triangleleft r) \cong \sum_{f: p(1) \rightarrow q(1)} \mathbf{Poly}\left(p \overset{f}{\curvearrowright} q, r\right). \quad (5.84)$$

Thus the functor  $(q \triangleleft -): \mathbf{Poly} \rightarrow \mathbf{Poly}$  is said to have a *left multiadjoint*.

$\diamond$

*Exercise 5.85.* In stating Proposition 5.79, we implicitly assumed that  $p \xrightarrow{f} q \in \mathbf{Poly}$  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  $f: p \rightarrow q \triangleleft 1$  and  $g: p' \rightarrow p$ , to what lens  $p' \xrightarrow{g \circ f} q \rightarrow p \xrightarrow{f} q$  should the covariant functor  $- \xrightarrow{f} q$  send  $g$ ? Prove that your construction is functorial.
2. Given lenses  $f: p \rightarrow q \triangleleft 1$  and  $h: q \rightarrow q'$ , to what lens  $p \xrightarrow{f \circ (h \triangleleft 1)} q' \rightarrow p \xrightarrow{f} q$  should the contravariant functor  $p \xrightarrow{f} -$  send  $h$ ? Prove that your construction is functorial.  $\diamond$

**Theorem 5.86** (Preservation of connected limits). The operation  $\triangleleft$  preserves connected limits on both sides. That is, if  $\mathcal{G}$  is a connected category,  $p: \mathcal{G} \rightarrow \mathbf{Poly}$  is a functor, and  $q \in \mathbf{Poly}$  is a polynomial, then there are natural isomorphisms

$$\left( \lim_{j \in \mathcal{G}} p_j \right) \triangleleft q \cong \lim_{j \in \mathcal{G}} (p_j \triangleleft q) \quad \text{and} \quad q \triangleleft \left( \lim_{j \in \mathcal{G}} p_j \right) \cong \lim_{j \in \mathcal{G}} (q \triangleleft p_j)$$

*Proof.* The claim for the left side is just a special case of Proposition 5.74; it remains to prove the claim on the right.

By Theorem 4.32,  $\mathbf{Poly}$  is complete, so by [nLa19, Theorem 4.3], it suffices to show that the functor  $(q \triangleleft -): \mathbf{Poly} \rightarrow \mathbf{Poly}$  preserves wide pullbacks on the right. By Proposition 5.79, the functor  $(q \triangleleft -): \mathbf{Poly} \rightarrow \mathbf{Poly}/q \triangleleft 1$  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  $q \triangleleft r \rightarrow q \triangleleft 1$  in  $\mathbf{Poly}/q \triangleleft 1$ , corresponding to a limit in  $\mathbf{Poly}$  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  $\mathbf{Poly}$  over  $q \triangleleft r$ , namely  $(q \triangleleft -): \mathbf{Poly} \rightarrow \mathbf{Poly}$  applied to the original wide pullback. So  $\triangleleft$  preserves wide pullbacks on the right.  $\square$

*Exercise 5.87.* Use Theorem 5.86 in the following.

1. Let  $p$  be a polynomial, thought of as a functor  $p: \mathbf{Set} \rightarrow \mathbf{Set}$ . Show that  $p$  preserves connected limits (of sets).
2. Show that for any polynomials  $p, q, r$  we have an isomorphism:

$$p \triangleleft (qr) \cong (p \triangleleft q) \times_{(p \triangleleft 1)} (p \triangleleft r). \quad (5.88)$$

3. Take the polynomials  $p, q, r$  from the counterexample you found in Exercise 5.62 #1 and check that (5.88) holds.  $\diamond$

While we're here, it will be helpful to record the following.

**Proposition 5.89.** For any polynomial  $q \in \mathbf{Poly}$ , tensoring with  $q$  (on either side) preserves connected limits. That is, if  $\mathcal{G}$  is connected and  $p: \mathcal{G} \rightarrow \mathbf{Poly}$  is a functor, then there is a natural isomorphism:

$$\left( \lim_{j \in \mathcal{G}} p_j \right) \otimes q \cong \lim_{j \in \mathcal{G}} (p_j \otimes q).$$

### 5.3.4 Interaction with parallel products

Before we get into how  $\otimes$  interacts with  $\triangleleft$ , here is a warm-up exercise.

*Exercise 5.90.* Let  $A$  and  $B$  be arbitrary sets, and let  $p$  be an arbitrary polynomial. Which of the following isomorphisms always hold?

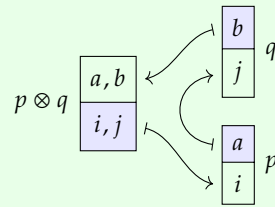
If the isomorphism does not always hold, is there still a canonical lens in one direction or the other?

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 5.91* (Lenses from  $\otimes$  to  $\triangleleft$ ). For any  $p$  and  $q$ , there is an interesting cartesian lens  $o_{p,q}: p \otimes q \rightarrow p \triangleleft q$  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  $p \otimes q \rightarrow q \triangleleft p$ .

Both  $\otimes$  and  $\triangleleft$  have the same monoidal unit, the identity functor  $y$ , whose identity is the unique lens  $y \rightarrow y$ . In fact the lenses  $o_{p,q}$  constitute a lax monoidal functor  $(\mathbf{Poly}, y, \otimes) \rightarrow (\mathbf{Poly}, y, \triangleleft)$ . In particular,  $o_{p,q}$  commutes with associators and unitors.

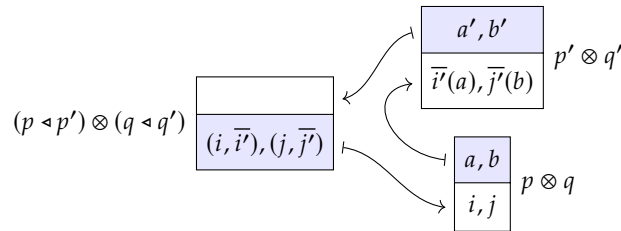
This can be used in the following way. Lenses  $p \rightarrow q \triangleleft r$  into composites are fairly easy to understand (through polyboxes, for example), whereas lenses  $q \triangleleft r \rightarrow p$  are not so easy to think about. However, given such a lens, one may always compose it with  $o_{q,r}$  to obtain a lens  $q \otimes r \rightarrow p$ . This is quite a bit simpler to think about: they are our familiar interaction patterns from Section 3.4.

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 map

$$(p \triangleleft p') \otimes (q \triangleleft q') \rightarrow (p \otimes q) \triangleleft (p' \otimes q'). \quad (5.92)$$

**Proposition 5.93.** The monoidal structures  $(y, \otimes)$  and  $(y, \triangleleft)$  together comprise a duoidal structure on  $\mathbf{Poly}$ .

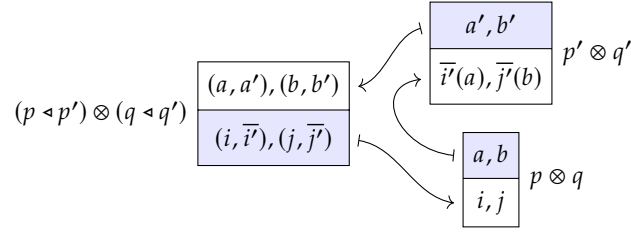
*Idea of proof.* The key is to give the natural lens from (5.92), as follows. A position of  $p \triangleleft p'$  is a pair  $(i, \bar{i}')$  with  $i \in p(1)$  and  $\bar{i}': p[i] \rightarrow p'(1)$ ; similarly, a position  $q \triangleleft q'$  is a pair  $(j, \bar{j}')$  with  $j \in q(1)$  and  $\bar{j}': q[j] \rightarrow q'(1)$ . 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  $(i, \bar{i}')$ , which can be given by the  $p[i]$ -direction  $a$  followed by a  $p'[\bar{i}'(a)]$ -direction, namely  $a'$ . We

also need a direction of  $q \triangleleft q'$ , which can be obtained analogously:



□

### 5.3.5 Interaction with vertical and cartesian lenses

We conclude this section by examining how  $\triangleleft$  interacts with vertical and cartesian lenses, as defined in Definition 4.50.

**Proposition 5.94** (Preservation of cartesian lenses). If  $\varphi: p \rightarrow p'$  and  $\psi: q \rightarrow q'$  are cartesian lenses, then so is  $\varphi \triangleleft \psi: p \triangleleft q \rightarrow p' \triangleleft q'$ .

*Proof.* We use the third characterization of cartesian lenses given in Proposition 4.58, as lenses whose naturality squares are pullbacks. For any sets  $A, B$  and function  $h: A \rightarrow B$ , consider the diagram

$$\begin{array}{ccccc} p \triangleleft q \triangleleft A & \rightarrow & p' \triangleleft q \triangleleft A & \rightarrow & p' \triangleleft q' \triangleleft A \\ \downarrow & & \downarrow & & \downarrow \\ p \triangleleft q \triangleleft B & \rightarrow & p' \triangleleft q \triangleleft B & \rightarrow & p' \triangleleft q' \triangleleft B. \end{array}$$

The square on the left is a pullback because  $\varphi: p \rightarrow p'$  is cartesian. Meanwhile, the square on the right is a pullback because  $\psi: q \rightarrow q'$  is cartesian and  $\triangleleft$  preserves pullbacks by Theorem 5.86. Hence the outer rectangle is a pullback as well, implying that  $\varphi \triangleleft \psi: p \triangleleft q \rightarrow p' \triangleleft q'$  is cartesian. □

*Exercise 5.95.* Let  $\varphi: p \rightarrow p'$  and  $\psi: q \rightarrow q'$  be lenses.

1. Show that if  $\varphi$  is an isomorphism and  $\psi$  is vertical, then  $\varphi \triangleleft \psi$  is vertical.
2. Find a vertical lens  $\varphi$  and a polynomial  $q$  for which  $\varphi \triangleleft q: p \triangleleft q \rightarrow p' \triangleleft q$  is not vertical. ◇

## 5.4 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 dependent types, e.g. as a sum-product-sum-product



$\Sigma \Pi \Sigma \Pi$ , which can be reduced to a single  $\Sigma \Pi$ ; in terms of trees, by stacking corollas on top of corollas; and in terms of polyboxes. We paid particular attention to maps into a composite  $p \rightarrow q \triangleleft r$ .

This allowed us to explain how to think of dynamical systems  $Sy^S \rightarrow p \triangleleft q$  with composite interfaces as multi-step machines: each state produces a  $p$ -output, and then for every  $p$ -input produces a  $q$ -output, and then for every  $q$ -input returns an updated  $S$ -state.

Finally, we discussed some facts of the composition product. For example, we showed that  $- \triangleleft q$  has a left adjoint  $\left[ \begin{smallmatrix} q \\ - \end{smallmatrix} \right]$  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 map  $\triangleleft \otimes \triangleleft \rightarrow \otimes \triangleleft \otimes$ , and how  $\triangleleft$  interacts with cartesian maps.

Polynomial composition 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 5.63) from Josh Meyers, though it may have already been known in the containers community.

## 5.5 Exercise solutions

*Solution to Exercise 5.8.*

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  $p[1] \rightarrow q(1)$ .
5. When  $\bar{j}_1: p[1] \rightarrow q(1)$  is one of the two possible bijections, we have that

$$\sum_{a \in p[1]} q[\bar{j}_1(a)] \cong q[1] + q[2] \cong 3 + 0 \cong 3.$$

When  $\bar{j}_1: p[1] \rightarrow q(1)$  sends everything to  $1 \in q(1)$ , we have that

$$\sum_{a \in p[1]} q[\bar{j}_1(a)] \cong q[1] + q[1] \cong 3 + 3 \cong 6.$$

Finally, when  $\bar{j}_1: p[1] \rightarrow q(1)$  sends everything to  $2 \in q(1)$ , we have that

$$\sum_{a \in p[1]} q[\bar{j}_1(a)] \cong q[2] + q[2] \cong 0 + 0 \cong 0.$$

6. Since  $p[2] \cong 1$  and  $q(1) \cong 2$ , there are  $2^1 = 2$  functions  $p[2] \rightarrow q(1)$ .
7. When  $j_2: p[2] \rightarrow q(1)$  maps to  $1 \in q(1)$ , we have that  $\sum_{a \in p[2]} q[\bar{j}_2(a)] \cong q[1] \cong 3$ , and when  $\bar{j}_2: p[2] \rightarrow q(1)$  maps to  $2 \in q(1)$ , we have that  $\sum_{a \in p[2]} q[\bar{j}_2(a)] \cong q[2] \cong 0$ .
8. From the previous parts, we have that

$$\sum_{i \in p(1)} \sum_{\bar{j}: p[i] \rightarrow q(1)} y^{\sum_{a \in p[i]} q[\bar{j}(a)]} \cong (2y^3 + y^6 + y^0) + (y^3 + y^0) \cong y^6 + 3y^3 + 2,$$

which agrees with what  $p \triangleleft q$  should be.

Solution to Exercise 5.9.

1. Given representable polynomials  $p := y^A$  and  $q := y^B$ , we have that  $p \triangleleft q \cong (y^B)^A \cong y^{AB}$ , which is also representable.
2. Given linear polynomials  $p := Ay$  and  $q := By$ , we have that  $p \triangleleft q \cong A(By) \cong AB y$ , which is also linear.
3. Given constant polynomials  $p := A$  and  $q := B$ , we have that  $p \triangleleft q \cong A$ , which is also constant (see also Exercise 5.28).

Solution to Exercise 5.10.

Given  $A \in \mathbf{Set}$  and  $q \in \mathbf{Poly}$ , we have

$$y^A \triangleleft q \cong \sum_{\bar{j}: A \rightarrow q(1)} y^{\sum_{a \in A} q[\bar{j}(a)]} \quad (5.7)$$

$$\begin{aligned} &\cong \sum_{\varphi: Ay \rightarrow q} y^{\sum_{a \in A} q[\varphi_1(a)]} \\ &\cong [Ay, q], \end{aligned} \quad (3.82)$$

for a lens  $\varphi: Ay \rightarrow q$  has an on-positions function  $A \rightarrow q(1)$  and uniquely determined on-directions functions.

Solution to Exercise 5.13.

We wish to show that (5.14) could replace (5.12) in Definition 5.11. We claim that (5.12) and (5.14) are in fact the same function; that is, that the following square commutes:

$$\begin{array}{ccc} p(q(X)) & \xrightarrow{f_{q(X)}} & p'(q(X)) \\ p(g_X) \downarrow & & \downarrow p'(g_X) \\ p(q'(X)) & \xrightarrow{f_{q'(X)}} & p'(q'(X)) \end{array}$$

Indeed it does, by the naturality of  $f$ .

Solution to Exercise 5.18.

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 5.19.

We have lenses  $f: p \rightarrow p'$  and  $g: q \rightarrow q'$ .

1. By Proposition 2.71, the  $q(X)$ -component of  $f$  is a function  $f_{q(X)}: p(q(X)) \rightarrow p'(q(X))$  that sends every  $(i, h)$  with  $i \in p(1)$  and  $h: p[i] \rightarrow q(X)$  to  $(f_1(i), f_i^\# \circ h)$ . We can think of the function

$h: p[i] \rightarrow q(X)$  equivalently as a function  $\bar{j}_i: p[i] \rightarrow q(1)$  and, for each  $a \in p[i]$ , a function  $h_a: q[\bar{j}_i(a)] \rightarrow X$ . So  $f_{q(X)}: (p \triangleleft q)(X) \rightarrow (p' \triangleleft q)(X)$  sends

$$(i, \bar{j}_i, (h_a)_{a \in p[i]}) \mapsto \left( f_1(i), f_i^\# \circ \bar{j}_i, \left( h_{f_i^\#(a')} \right)_{a' \in p'[f_1(i)]} \right).$$

2. By Proposition 2.71, the  $X$ -component of  $g$  is a function  $g_X: q(X) \rightarrow q'(X)$  that sends every  $(j, k)$  with  $j \in q(1)$  and  $k: q[j] \rightarrow X$  to  $(g_1(j), g_j^\# \circ k)$  in  $q'(X)$ . Then by Proposition 2.47, applying  $p'$  to this  $X$ -component yields a function  $p'(q(X)) \rightarrow p'(q'(X))$  that sends every  $(i', \bar{j}'_{i'}, (h'_{a'})_{a' \in p'[i']})$  with  $i' \in p'(1)$  as well as  $\bar{j}'_{i'}: p'[i'] \rightarrow q(1)$  and  $h'_{a'}: q[\bar{j}'_{i'}(a')] \rightarrow X$  to

$$\left( i', \bar{j}'_{i'} \circ g_1, \left( g_{\bar{j}'_{i'}(a')}^\# \circ h'_{a'} \right)_{a' \in p'[i']} \right).$$

3. By Definition 5.11, the horizontal composite of  $f$  and  $g$  is the natural transformation  $f \triangleleft g: p \triangleleft p' \rightarrow q \triangleleft q'$  whose  $X$ -component is the composite of the answers to #1 and #2, sending

$$\begin{aligned} (i, \bar{j}_i, (h_a)_{a \in p[i]}) &\mapsto \left( f_1(i), f_i^\# \circ \bar{j}_i, \left( h_{f_i^\#(a')} \right)_{a' \in p'[f_1(i)]} \right) \\ &\mapsto \left( f_1(i), f_i^\# \circ \bar{j}_i \circ g_1, \left( g_{\bar{j}_i(f_i^\#(a'))}^\# \circ h_{f_i^\#(a')} \right)_{a' \in p'[f_1(i)]} \right). \end{aligned}$$

4. We use Corollary 2.73 to translate the answer to #3 into a lens  $f \triangleleft g: p \triangleleft q \rightarrow p' \triangleleft q'$ , 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  $\bar{j}_i: p[i] \rightarrow q(1)$  to

$$(f_1(i), f_i^\# \circ \bar{j}_i \circ g_1).$$

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  $\iota_d: q[\bar{j}_i(a)] \rightarrow (p \triangleleft q)[(i, \bar{j}_i)] \cong \sum_{a \in p[i]} q[\bar{j}_i(a)]$  is the canonical inclusion, then take the last coordinate of the result, we obtain for each  $a' \in p'[f_1(i)]$  the function

$$q'[g_1(\bar{j}_i(f_i^\#(a')))] \xrightarrow{g_{\bar{j}_i(f_i^\#(a'))}^\#} q[\bar{j}_i(f_i^\#(a'))] \xrightarrow{\iota_{f_i^\#(a')}} \sum_{a \in p[i]} q[\bar{j}_i(a)] \cong (p \triangleleft q)[(i, \bar{j}_i)].$$

These can equivalently be thought of as a single function from

$$\sum_{a' \in p'[f_1(i)]} q'[g_1(\bar{j}_i(f_i^\#(a')))] \cong (p' \triangleleft q')[(f \triangleleft g)_1(i, \bar{j}_i)]$$

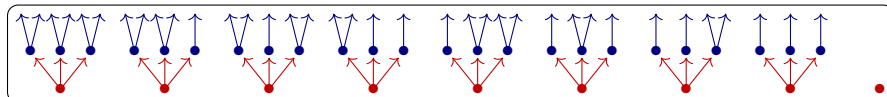
which Corollary 2.73 tells us is the on-directions function of  $f \triangleleft g$  at  $(i, \bar{j}_i)$ , that sends every  $(a', b')$  with  $a' \in p'[f_1(i)]$  and  $b' \in q'[g_1(\bar{j}_i(f_i^\#(a')))]$  to

$$\left( f_i^\#(a'), g_{\bar{j}_i(f_i^\#(a'))}^\#(b') \right).$$

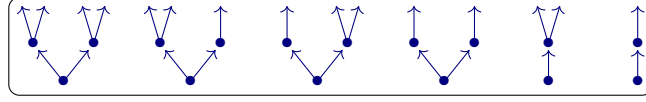
*Solution to Exercise 5.26.*

We have  $p := y^2 + y$  and  $q := y^3 + 1$  as in (5.22).

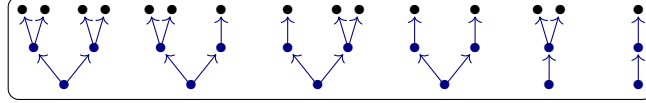
1. Here is a picture of  $q \triangleleft p$ , where each tree is obtained by taking a  $q$ -corolla and gluing  $p$ -corollas to every leaf:



2. Here is a picture of  $p \triangleleft p$ :



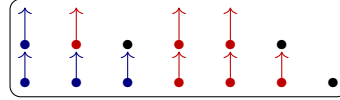
3. To obtain a picture of  $p \triangleleft p \triangleleft 1$ , we take our picture of  $p \triangleleft p$  and glue the single, leafless 1-root to every (height-2) leaf:



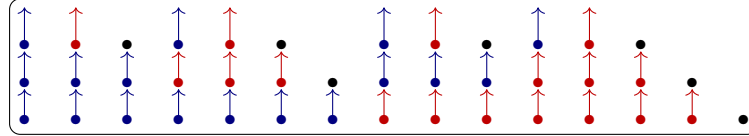
Now  $r := 2y + 1$ . Before we draw the composites, here's a picture of  $r$  itself, with different colors to distinguish the different positions:



4. Here is a picture of  $r \triangleleft r$ :

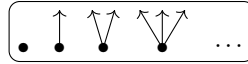


5. Here is a picture of  $r \triangleleft r \triangleleft r$ :



*Solution to Exercise 5.28.*

1. We pick the list polynomial,  $p := 1 + y + y^2 + y^3 + \dots$ , drawn as follows:



Then here is a picture of  $p \triangleleft 1$ :



Below,  $X$  is a set and  $p$  is a polynomial.

2. A constant functor composed with any functor is still the same constant functor, so  $X \triangleleft p \cong X$ . We can also verify this using (5.6):

$$X \triangleleft p \cong \sum_{i \in X} \prod_{a \in \emptyset} \sum_{j \in p(1)} \prod_{b \in p[j]} y \cong \sum_{i \in X} 1 \cong X.$$

3. When viewed as functors, it is easy to see that  $p \triangleleft X \cong p(X)$ . We can also verify this using (5.6):

$$p \triangleleft X \cong \sum_{i \in p(1)} \prod_{a \in p[i]} \sum_{j \in X} \prod_{b \in \emptyset} y \cong \sum_{i \in p(1)} \prod_{a \in p[i]} \sum_{j \in X} 1 \cong \sum_{i \in p(1)} \prod_{a \in p[i]} X \cong \sum_{i \in p(1)} X^{p[i]} \cong p(X).$$

*Solution to Exercise 5.29.*

Let  $\varphi: p \rightarrow q$  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  $\text{id}_X: X \rightarrow X$

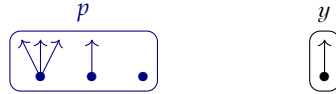
on  $X$  as a set. Then by Definition 5.11, any component of the composition product  $\varphi \triangleleft X$  viewed as a natural transformation is given by the composite function

$$p(X) \xrightarrow{\varphi_X} q(X) \xrightarrow{q(\text{id}_X)} q(X).$$

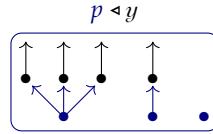
By functoriality,  $q(\text{id}_X)$  is itself an identity function, so every component of  $\varphi \triangleleft X$  is the  $X$ -component of  $\varphi$ . Therefore  $\varphi \triangleleft X$  as a function can be identified with the  $X$ -component of  $\varphi$ , as desired.

*Solution to Exercise 5.30.*

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:

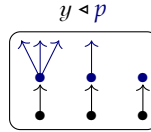


Now here is a picture of  $p \triangleleft y$ , obtained by gluing the one-leaf  $y$ -corolla to all the leaves of each  $p$ -corolla in turn:



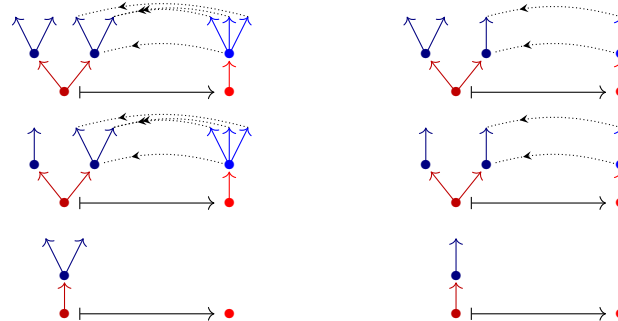
This is just  $p$  with every direction extended up one level, so it is still a picture of  $p$ .

And here is a picture of  $y \triangleleft p$ , obtained by gluing each  $p$ -corolla to the single leaf of  $y$ :



*Solution to Exercise 5.32.*

Using the definitions, instructions, and style from Example 5.31, we draw  $\psi \triangleleft \varphi: q \triangleleft p \rightarrow q' \triangleleft p'$ :



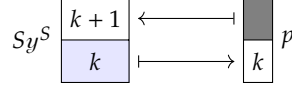
*Solution to Exercise 5.33.*

Given arbitrary polynomials  $p, q, r$  and lenses  $\varphi: q \rightarrow p \triangleleft q$  and  $\psi: q \rightarrow q \triangleleft r$ , it is *not* necessarily the case that  $\varphi \circ (p \triangleleft \psi) = \psi \circ (\varphi \triangleleft r)$ ! After all, we can let  $p := y$  and  $q := 2$  so that  $\varphi$  is a lens  $2 \rightarrow y \triangleleft 2 \cong 2$  (see Exercise 5.30) and  $\psi$  is a lens  $2 \rightarrow 2 \triangleleft r \cong 2$  (see Exercise 5.28). Then by following the instructions for interpreting a composition product of lenses from either Exercise 5.19 or Example 5.31, we can verify that  $p \triangleleft \psi = y \triangleleft \psi$  is a lens  $2 \cong y \triangleleft 2 \rightarrow y \triangleleft 2 \triangleleft r \cong 2$  equivalent to the lens  $\psi$ , while  $\varphi \triangleleft r$  is a lens  $2 \cong 2 \triangleleft r \rightarrow y \triangleleft 2 \triangleleft r \cong 2$  equivalent to the lens  $\varphi$ . If, say, we let  $\varphi: 2 \rightarrow 2$  be the function sending everything to  $1 \in 2$  and  $\psi: 2 \rightarrow 2$  be the function sending everything to  $2 \in 2$ , then in this case  $\varphi \circ (p \triangleleft \psi) = \varphi \circ \psi \neq \psi \circ \varphi = \psi \circ (\varphi \triangleleft r)$ .

Solution to Exercise 5.51.

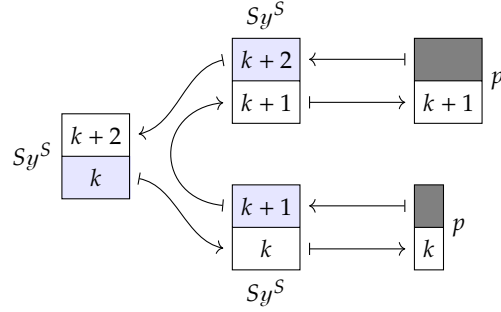
Given  $S := \mathbb{N}$  and  $p := \mathbb{R}y^1$ , we have a dynamical system  $\varphi: Sy^S \rightarrow p$  given by  $\varphi_1(k) := k$  and  $\varphi_k^\#(1) := k + 1$ .

- Here is the polybox picture for  $\varphi$  (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.

- Here is the polybox picture for  $\delta \circ (\varphi \triangleleft \varphi): Sy^S \rightarrow p \triangleleft p$ :



The new system has  $p \triangleleft p \cong \mathbb{R}y \triangleleft \mathbb{R}y \cong \mathbb{R}^2y$  as its interface. Indeed, we see that it returns two numbers at once: the current state  $k$  (what the first run through  $\varphi$  would return) as well as the increased state  $k + 1$  (what the second run through  $\varphi$  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 5.52.

We give two reasonable (and of course equivalent) ways to interpret the transition lens  $\delta: Sy^S \rightarrow Sy^S \triangleleft Sy^S$ .

One way is to first evaluate its interface as  $Sy^S \triangleleft Sy^S \cong S(Sy^S)^S \cong (S \times S^S)y^{S \times S}$ . 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  $S \rightarrow S$ , 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 5.47 to interpret  $\delta$  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 5.58.

To prove Proposition 5.53, it suffices to verify (5.56) and (5.57), as (5.54) and (5.55) 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 (5.6), we have that

$$\begin{aligned} \left( \sum_{a \in A} q_a \right) \triangleleft r &\cong \sum_{\substack{a \in A, \\ j \in q_a(1)}} \prod_{b \in q_a[j]} \sum_{k \in r(1)} \prod_{c \in r[k]} y \\ &\cong \sum_{a \in A} \sum_{j \in q_a(1)} \prod_{b \in q_a[j]} \sum_{k \in r(1)} \prod_{c \in r[k]} y \end{aligned}$$

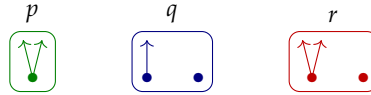
$$\cong \sum_{a \in A} (q_a \triangleleft r).$$

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  $\bar{j}: (a \in A) \rightarrow q_a(1)$  is  $\sum_{a \in A} q_a[\bar{j}(a)]$ . So by (5.6), we have that

$$\begin{aligned} \left( \prod_{a \in A} q_a \right) \triangleleft r &\cong \sum_{\bar{j} \in \prod_{a \in A} q_a(1)} \prod_{\substack{a \in A, \\ b \in q_a[\bar{j}(a)]}} \sum_{k \in r(1)} \prod_{c \in r[k]} y \\ &\cong \prod_{a \in A} \sum_{j \in q_a(1)} \prod_{b \in q_a[j]} \sum_{k \in r(1)} \prod_{c \in r[k]} y \\ &\cong \prod_{a \in A} (q_a \triangleleft r). \end{aligned} \tag{2.29}$$

*Solution to Exercise 5.60.*

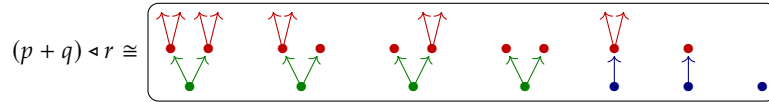
We want an intuitive understanding of the left distributivity of  $\triangleleft$  over  $+$ . Let  $p := y^2$ ,  $q := y + 1$ , and  $r := y^2 + 1$ , as shown here:



Then  $p + q \cong y^2 + y + 1$  can be drawn as follows, consisting of every  $p$ -corolla and every  $q$ -corolla:

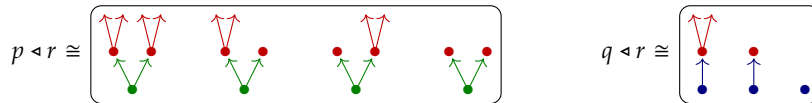


We can therefore draw  $(p + q) \triangleleft r$  by gluing  $r$ -corollas to leaves of  $p + q$  in every way, as follows:

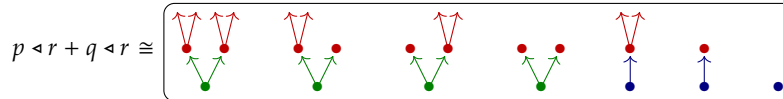


So each tree in  $(p + q) \triangleleft r$  is obtained by taking either a  $p$ -corolla or a  $q$ -corolla, then attaching an  $r$ -corolla to each leaf.

Alternatively, we can compute  $p \triangleleft r$  and  $q \triangleleft r$  separately, gluing  $r$ -corollas to leaves of  $p$  in every way, then to leaves of  $q$  in every way:



Their coproduct then consists of all the trees from  $p \triangleleft r$  and all the trees from  $q \triangleleft r$ :



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 glue  $r$ -corollas to leaves first, or if we pool together corollas from  $p$  and  $q$  first—the processes are equivalent. Hence the isomorphism  $(p + q) \triangleleft r \cong p \triangleleft r + q \triangleleft r$  holds.

*Solution to Exercise 5.61.*

Given a set  $A$  and polynomials  $p, q$ , the left distributivity of  $\triangleleft$  over products from (5.57) implies that  $(Ap) \triangleleft q \cong (A \triangleleft q)(p \triangleleft q)$ , while Exercise 5.28 #3 implies that  $A \triangleleft q \cong A$ . So  $(Ap) \triangleleft q \cong A(p \triangleleft q)$ .

*Solution to Exercise 5.62.*

1. Let  $p := y + 1$ ,  $q := 1$ , and  $r := 0$ . Then  $p \triangleleft (qr) \cong (y + 1) \triangleleft 0 \cong 1$ , while  $(p \triangleleft q)(p \triangleleft r) \cong ((y + 1) \triangleleft 1)((y + 1) \triangleleft 0) \cong 2 \times 1 \cong 2$ .
2. Again let  $p := y + 1$ ,  $q := 1$ , and  $r := 0$ . Then  $p \triangleleft (q + r) \cong (y + 1) \triangleleft 1 \cong 2$ , while  $(p \triangleleft q) + (p \triangleleft r) \cong ((y + 1) \triangleleft 1) + ((y + 1) \triangleleft 0) \cong 2 + 1 \cong 3$ .

*Solution to Exercise 5.66.*

We prove Proposition 5.63 by observing that

$$\mathbf{Poly}(p, r \triangleleft q) \cong \prod_{i \in p(1)} r(q(p[i])) \quad (2.55)$$

$$\cong \mathbf{Poly}\left(\sum_{i \in p(1)} y^{q(p[i])}, r\right) \quad (2.55)$$

$$\cong \mathbf{Poly}\left(\begin{bmatrix} q \\ p \end{bmatrix}, r\right). \quad (5.65)$$

*Solution to Exercise 5.68.*

1. Given a polynomial  $q$  and a lens  $\varphi: p \rightarrow p'$ , the functor  $\begin{bmatrix} q \\ \varphi \end{bmatrix}$  should send  $\varphi$  to a lens  $\begin{bmatrix} q \\ p \end{bmatrix} \rightarrow \begin{bmatrix} q \\ p' \end{bmatrix}$ ; by (5.65), this is a lens

$$\begin{bmatrix} q \\ \varphi \end{bmatrix}: \sum_{i \in p(1)} y^{q(p[i])} \rightarrow \sum_{i' \in p'(1)} y^{q(p'[i'])}.$$

We give  $\begin{bmatrix} q \\ \varphi \end{bmatrix}$  the same on-positions function  $p(1) \rightarrow p'(1)$  that  $\varphi$  has. Then viewing  $q$  as a functor, we define the on-directions function  $\begin{bmatrix} q \\ \varphi \end{bmatrix}_i^\# : q(p'[\varphi_1(i)]) \rightarrow q(p[i])$  for each  $i \in p(1)$  to be the function obtained by applying  $q$  to the corresponding on-directions function  $\varphi_i^\# : p'[\varphi_1(i)] \rightarrow p[i]$ . Functoriality follows trivially on positions and by the functoriality of  $q$  itself on directions.

2. Given a polynomial  $p$  and a lens  $\psi: q' \rightarrow q$ , the functor  $\begin{bmatrix} \psi \\ p \end{bmatrix}$  should send  $\psi$  to a lens  $\begin{bmatrix} q \\ p \end{bmatrix} \rightarrow \begin{bmatrix} q' \\ p \end{bmatrix}$ ; by (5.65), this is a lens

$$\begin{bmatrix} \psi \\ p \end{bmatrix}: \sum_{i \in p(1)} y^{q(p[i])} \rightarrow \sum_{i \in p(1)} y^{q'(p[i])}.$$

We let the on-positions function of  $\begin{bmatrix} \psi \\ p \end{bmatrix}$  be the identity on  $p(1)$ . Then viewing  $\psi$  as a natural transformation, we define the on-directions function  $\begin{bmatrix} \psi \\ p \end{bmatrix}_i^\# : q'(p[i]) \rightarrow q(p[i])$  for each  $i \in p(1)$  to be the  $p[i]$ -component of  $\psi$ . Functoriality follows trivially on positions and because natural transformations compose componentwise on directions.

*Solution to Exercise 5.3.2.*

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  $p \rightarrow \begin{bmatrix} q \\ p \end{bmatrix} \triangleleft q$ , and second show that it is universal. But this is exactly the content of Proposition 5.63: the adjunction (5.64) provides a unit  $p \rightarrow \begin{bmatrix} q \\ p \end{bmatrix} \triangleleft q$  that is universal in the appropriate way.

*Solution to Exercise 5.70.*

We are given  $A, B \in \mathbf{Set}$  and  $p, q \in \mathbf{Poly}$

1. By (5.65),

$$\begin{bmatrix} B \\ A \end{bmatrix} = \sum_{i \in A} y^B \cong Ay^B,$$

so by (5.64),

$$\mathbf{Poly}(Ay^B, p) \cong \mathbf{Poly}(A, p \triangleleft B).$$

But  $A$  and  $p \triangleleft B \cong p(B)$  are both constants, so a lens  $A \rightarrow p \triangleleft B$  is just a function  $A \rightarrow p(B)$  on positions. Hence (5.71) follows.



2. We prove (5.72) in two parts: that

$$\mathbf{Poly}(Ay \triangleleft p, q) \cong \mathbf{Poly}(p, y^A \triangleleft q) \quad (5.96)$$

and that

$$\mathbf{Poly}(p \triangleleft y^B, q) \cong \mathbf{Poly}(p, q \triangleleft By). \quad (5.97)$$

We have that  $Ay \triangleleft p \cong Ap$  and  $y^A \triangleleft q \cong q^A$ , so (5.96) follows from (4.9). Meanwhile, for (5.97), we have by (5.65) that

$$\begin{bmatrix} By \\ p \end{bmatrix} = \sum_{i \in p(1)} y^{Bp[i]} \cong p \triangleleft y^B,$$

so (5.72) follows from (5.64). Then combining (5.96) and (5.97) yields

$$\mathbf{Poly}(Ay \triangleleft p \triangleleft y^B, q) \cong \mathbf{Poly}(p \triangleleft y^B, y^A \triangleleft q) \cong \mathbf{Poly}(p, y^A \triangleleft q \triangleleft By).$$

*Solution to Exercise 5.76.*

1. We wish to solve Exercise 5.28 #3 using (5.75) and (5.56). If we set  $\mathcal{G}$  in (5.75) to be the empty category, then the limit of the functor from  $\mathcal{G}$  is just the terminal object. It follows that  $1 \triangleleft p \cong 1$ . In other words, since  $\triangleleft$  preserves limits on the left, and since terminal objects are limits,  $\triangleleft$  preserves terminal objects on the left.

Then a set  $X$  can be written as a sum  $\sum_{x \in X} 1$ , so by (5.56),

$$X \triangleleft p \cong \left( \sum_{x \in X} 1 \right) \triangleleft p \cong \sum_{x \in X} (1 \triangleleft p) \cong \sum_{x \in X} 1 \cong X.$$

2. If we set  $\mathcal{G}$  in (5.75) to be the discrete category on the set  $A$ , then the limit of a functor from  $\mathcal{G}$  is just an  $A$ -fold product, so (5.57) follows. In other words, since  $\triangleleft$  preserves limits on the left, and since products are limits,  $\triangleleft$  preserves products on the left.

*Solution to Exercise 5.83.*

1. Note that  $\mathbf{Poly}(p, q \triangleleft 1) \cong \mathbf{Set}(p(1), q(1))$ . A lens  $p \rightarrow q \triangleleft r$  is an element of

$$\prod_{i \in p(1)} \sum_{j \in q(1)} \prod_{b \in q[j]} \sum_{k \in r(1)} \prod_{c \in r[k]} p[i] \cong \sum_{f: p(1) \rightarrow q(1)} \prod_{i \in p(1)} \prod_{b \in q[j]} \sum_{k \in r(1)} \prod_{c \in r[k]} p[i]$$

Fixing the function  $f: p(1) \rightarrow q(1)$  as implicit in  $p$ , we get

$$\begin{aligned} \mathbf{Poly}/(q \triangleleft 1)(p, q \triangleleft r) &\cong \prod_{i \in p(1)} \prod_{b \in q[j]} \sum_{k \in r(1)} \prod_{c \in r[k]} p[i] \\ &\cong \mathbf{Poly}\left(p \overset{f}{\frown} q, r\right), \end{aligned}$$

2. This follows from the first isomorphism above.

*Solution to Exercise 5.85.*

1. Given lenses  $f: p \rightarrow q \triangleleft 1$  and  $g: p' \rightarrow p$ , the functor  $- \overset{f}{\frown} q$  should send  $g$  to a lens  $p' \overset{g \circ f}{\frown} q \rightarrow p \overset{f}{\frown} q$ ; by (5.81), this is a lens

$$g \overset{f}{\frown} q: \sum_{i' \in p'(1)} q[f_1(g_1(i'))]y^{p'[i']} \rightarrow \sum_{i \in p(1)} q[f_1(i)]y^{p[i]}.$$

We give  $g \overset{f}{\frown} q$  an on-positions function that is the on-positions function  $p'(1) \rightarrow p(1)$  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  $g_{i'}^\#: p[g_1(i')] \rightarrow p'[i']$ . Functoriality follows trivially on both positions and directions.

2. Given a polynomial  $p$  and lenses  $f: p \rightarrow q \triangleleft 1$  and  $h: q \rightarrow q'$ , the functor  $p \xrightarrow{f} -$  should send  $h$  to a lens  $p \xrightarrow{f \circ (h \triangleleft 1)} q' \rightarrow p \xrightarrow{f} q$ ; by (5.81), this is a lens

$$p \xrightarrow{f} h: \sum_{i \in p(1)} q'[h_1(f_1(i))]y^{p[i]} \rightarrow \sum_{i \in p(1)} q[f_1(i)]y^{p[i]}.$$

We let the on-positions function of  $p \xrightarrow{f} h$  be the identity on the first coordinate  $i \in p(1)$  and the on-directions function  $h_{f_1(i)}^\# : q'[h_1(f_1(i))] \rightarrow q[f_1(i)]$  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 5.87.*

1. Given a polynomial functor  $p: \mathbf{Set} \rightarrow \mathbf{Set}$ , we wish to show that  $p$  preserves connected limits of sets; that is, for a connected category  $\mathcal{G}$  and a functor  $X: \mathcal{G} \rightarrow \mathbf{Set}$ , we have

$$p\left(\lim_{j \in \mathcal{G}} X_j\right) \cong \lim_{j \in \mathcal{G}} p(X_j).$$

But we can identify  $\mathbf{Set}$  with the full subcategory of constant functors in  $\mathbf{Poly}$  and instead view  $X$  as a functor into  $\mathbf{Poly}$ . Then by Exercise 5.28 #3, the left hand side of the isomorphism we seek is isomorphic to  $p \triangleleft \left(\lim_{j \in \mathcal{G}} X_j\right)$ , while the right hand side is isomorphic to  $\lim_{j \in \mathcal{G}} (p \triangleleft X_j)$ . These are isomorphic by Theorem 5.86.

2. Given  $p, q, r \in \mathbf{Poly}$ , we wish to show that (5.88) holds. As 1 is terminal in  $\mathbf{Poly}$ , the product  $qr$  can also be written as the pullback  $q \times_1 r$ . While products are not connected limits, pullbacks are, so by Theorem 5.86, they are preserved by precomposition with  $p$ . Hence the desired isomorphism follows.
3. We'll show that (5.88) holds for  $p := y + 1$ ,  $q := 1$ , and  $r := 0$ . 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 4.37 that the position-set of a pullback in  $\mathbf{Poly}$  is just the pullback of the position-sets in  $\mathbf{Set}$ , while the direction-sets are given by a pushout of direction-sets in  $\mathbf{Set}$ . 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 5.90.*

Here  $A$  and  $B$  are sets and  $p$  is a polynomial.

1. The isomorphism always holds: we have that  $(Ay) \otimes (By) \cong AB y \cong (Ay) \triangleleft (By)$ .
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  $AB \rightarrow B$ ; but there is not always a canonical lens  $B \rightarrow AB$  (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  $y^A \triangleleft p \cong \sum_{i: A \rightarrow p(1)} y^{\sum_{a \in A} p[i(a)]}$ . So there is a lens from left to right whose on-positions function sends  $i \in p(1)$  to the constant function  $A \rightarrow p(1)$  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  $p \triangleleft By \cong \sum_{i \in p(1)} \sum_{\bar{b}: p[i] \rightarrow B} y^{\sum_{a \in p[i]} 1} \cong \sum_{i \in p(1)} B^{p[i]} y^{p[i]}$ . 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  $c_b: p[i] \rightarrow B$  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 5.95.*

Here  $\varphi: p \rightarrow p'$  and  $\psi: q \rightarrow q'$  are lenses.

1. If  $\varphi$  is an isomorphism and  $\psi$  is vertical, then  $\psi \triangleleft 1: q \triangleleft 1 \rightarrow q' \triangleleft 1$  is an isomorphism, so  $\varphi \triangleleft \psi \triangleleft 1: p \triangleleft q \triangleleft 1 \rightarrow p' \triangleleft q' \triangleleft 1$  is an isomorphism as well. Thus  $\varphi \triangleleft \psi$  is vertical.
2. If  $\varphi$  is the unique lens  $y \rightarrow 1$  and  $q = 0$ , then  $\varphi$  is vertical, but since  $y \triangleleft 0 \cong 0$  and  $1 \triangleleft 0 \cong 1$ , the lens  $\varphi \triangleleft 0: 0 \rightarrow 1$  is not.



## Chapter 6

---

# Polynomial comonoids and cofunctors

---

*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. Let's go through it.

### 6.1 State systems, categorically

Since defining dependent dynamical systems in Definition 3.17, 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 3.43, 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  $s \in \mathbf{Poly}$  and attempt to characterize what it means for  $s$  to be a state system using only the categorical machinery of **Poly**. We will continue to refer to the positions of  $s$  as *states*, but we will shift from thinking of the directions of  $s$  as states to thinking of them as transitions from one state to another.

### 6.1.1 The do-nothing enclosure

In Example 3.43, we saw that every state system  $\mathfrak{s}$  is equipped with a *do-nothing enclosure*: a lens  $\epsilon: \mathfrak{s} \rightarrow y$  that picks out a direction at each state that we would like to interpret as “doing nothing” and remaining at that state.

We drew  $\epsilon$  in polyboxes in Example 5.44, 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 enclosure 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  $\epsilon$  identifies: call it  $\text{id}_s$ , because it behaves like a sort of identity operation on the state  $s$ .

So the revised polyboxes for the do-nothing enclosure  $\epsilon: \mathfrak{s} \rightarrow y$  are as follows:

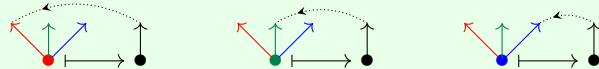
$$\begin{array}{|c|} \hline \text{id}_s \\ \hline s \\ \hline \end{array} \quad \epsilon \quad (6.1)$$

*Exercise 6.2.* Say I have a polynomial  $\mathfrak{s} \in \mathbf{Poly}$ , and I tell you that there is a lens  $\epsilon: \mathfrak{s} \rightarrow y$ . What can you say about the polynomial  $\mathfrak{s}$ ?  $\diamond$

*Example 6.3* (The do-nothing enclosure in tree pictures). We have seen the do-nothing enclosure drawn in polyboxes, but let’s see what it looks like in our tree pictures. We’ll take  $\mathfrak{s} := \{\bullet, \bullet, \bullet\}y^{\{\bullet, \bullet, \bullet\}}$ , drawn as follows:



Then the do-nothing enclosure  $\epsilon: \mathfrak{s} \rightarrow y$  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 enclosure on its own, so let us revisit the other lens that every state system is equipped with before considering the relationship between the two.

### 6.1.2 The transition lens

We saw in Example 5.50 that  $\mathfrak{s}$  also comes equipped with a *transition lens*: a lens  $\delta: \mathfrak{s} \rightarrow \mathfrak{s} \triangleleft \mathfrak{s}$ , 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,  $\text{tgt}(s_0, a_1)$  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  $\mathfrak{s}[s_0] \rightarrow \mathfrak{s}(1)$ ; 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  $s \rightarrow t$  denote the unique direction at  $s$  that leads to  $t$ , so that  $\text{tgt}(s, s \rightarrow t) = t$ . So we can redraw our transition lens as



(6.4)

In addition to the fact that  $\text{tgt}(s_0, s_0 \rightarrow s_1) = s_1$  as intended, this picture tells us two more properties of  $\delta$ .

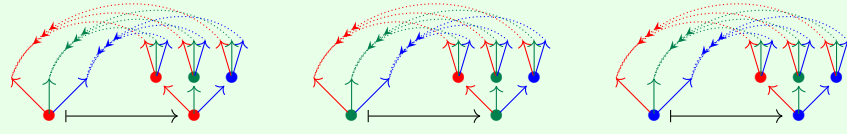
The first is that the bottom arrow is the identity on  $\mathfrak{s}(1)$ . This is something we would like to be able to express categorically in the language of **Poly**. We'll see that this property falls out naturally when we express how the transition lens plays nicely with the do-nothing enclosure in Section 6.1.3.

The second is that the *run* arrow, which runs the transition  $s_0 \rightarrow s_1$  and the transition  $s_1 \rightarrow s_2$  together into a transition starting at  $s_0$ , should have the same target as the 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

$$\text{tgt}(s_0, \text{run}(s_0, s_0 \rightarrow s_1, s_1 \rightarrow s_2)) = s_2 = \text{tgt}(\text{tgt}(s_0, s_0 \rightarrow s_1), s_1 \rightarrow s_2). \quad (6.5)$$

We'll see that this property arises naturally when we generalize the transition lens to more than two steps in Section 6.1.4.

*Example 6.6* (The transition lens in tree pictures). Continuing from Example 6.3, we draw the transition lens  $\delta: \mathfrak{s} \rightarrow \mathfrak{s} \triangleleft \mathfrak{s}$  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 gluing more  $\mathfrak{s}$ -corollas to each of its leaves:



On positions, the target function of  $\delta$  tells us which root of  $\mathfrak{s}$  to glue to each leaf of  $\mathfrak{s}$ . Then on directions, the run function of  $\delta$  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  $\delta$  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 (6.5), 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.

### 6.1.3 The do-nothing enclosure coheres with the transition lens

For each state  $s \in \mathfrak{s}(1)$ , the do-nothing enclosure  $\epsilon: \mathfrak{s} \rightarrow y$  picks out the  $\mathfrak{s}[s]$ -direction  $\text{id}_s$  that “does nothing” and keeps the system in the same state  $s$ . But it is the transition lens  $\delta: \mathfrak{s} \rightarrow \mathfrak{s} \triangleleft \mathfrak{s}$  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  $\text{id}_s$ , so let’s try to figure out what should happen when we set each one in turn to  $\text{id}_s$ .

We can draw in polyboxes what happens when we set  $\text{id}_s$ , as specified by  $\epsilon$ , to be



the first direction that  $\delta$  runs together like this:



Reading this picture from left to right, we see that it depicts the polyboxes of the composite lens  $\delta \circ (\epsilon \triangleleft s) : s \rightarrow y \triangleleft s \cong s$  (recall that we sometimes denote the identity lens on  $s$  also by  $s$ ). 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):

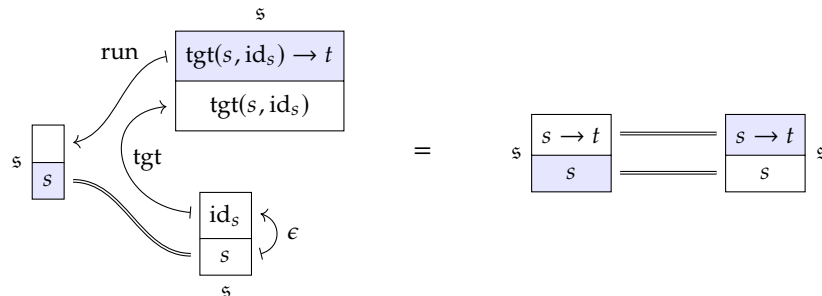


Now what should  $\text{tgt}(\text{id}_s)$  be, and what should go in the direction box on the left?

If following the direction  $\text{id}_s$  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  $\text{id}_s$  with any other direction  $s \rightarrow t$  from  $s$  should be no different from the direction  $s \rightarrow t$  on its own. So

$$\text{tgt}(s, \text{id}_s) = s \quad \text{and} \quad \text{run}(s, \text{id}_s, s \rightarrow t) = s \rightarrow t.$$

In fact,  $\text{id}_s$  should really just be the direction  $s \rightarrow s$ . Pictorially, we have the equation



Or, if you prefer, we might say that  $\delta \circ (\epsilon \triangleleft s) = \text{id}_s$ , or that the following diagram commutes:

$$\begin{array}{ccc}
 y \triangleleft s & \xlongequal{\quad} & s \\
 \epsilon \triangleleft s \swarrow & & \downarrow \delta \\
 & & s \triangleleft s
 \end{array}$$

This commutative diagram captures one way in which  $\epsilon$  and  $\delta$  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  $\delta$  runs together to what is specified by  $\epsilon$ , rather than the first? To answer this, we should look at the composite lens  $\delta \circ (s \triangleleft \epsilon) : s \rightarrow s \triangleleft y \cong s$  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  $s$ .

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  $\delta$  on the left must send  $s$  to itself: it is the identity function on  $s(1)$ . Indeed, this is exactly what we wanted to say about that arrow in Section 6.1.2.

Meanwhile, matching up directions in the domain tells us that

$$\text{run}(s, s \rightarrow t, \text{id}_t) = s \rightarrow t,$$

as we would expect:  $\text{id}_t$  is just be the direction  $t \rightarrow t$ .

More concisely, we can express both these facts in **Poly** via the equation  $\delta \circ (s \triangleleft \epsilon) = \text{id}_s$ . The corresponding commutative diagram is as follows:

$$\begin{array}{ccc}
 s & \xlongequal{\quad} & s \triangleleft y \\
 \delta \downarrow & \nearrow s \triangleleft \epsilon & \\
 s & \triangleleft & s
 \end{array}$$

We can combine this with our previous commutative diagram to say that the relationship between the do-nothing enclosure  $\epsilon : s \rightarrow y$  and the transition lens  $\delta : s \rightarrow s \triangleleft s$  of

a state system  $\mathfrak{s}$  is captured in **Poly** by the following commutative diagram:

$$\begin{array}{ccccc}
 y \triangleleft \mathfrak{s} & \xlongequal{\quad} & \mathfrak{s} & \xlongequal{\quad} & \mathfrak{s} \triangleleft y \\
 \swarrow \epsilon \triangleleft \mathfrak{s} & & \downarrow \delta & & \searrow \mathfrak{s} \triangleleft \epsilon \\
 & & \mathfrak{s} \triangleleft \mathfrak{s} & & 
 \end{array} \tag{6.7}$$

#### 6.1.4 The transition lens is coassociative

Toward the end of Example 5.50, we noted that while the transition lens  $\delta: \mathfrak{s} \rightarrow \mathfrak{s} \triangleleft \mathfrak{s}$  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  $\mathfrak{s} \rightarrow \mathfrak{s}^{\triangleleft 3}$  that runs three directions together by taking either one of the composite lenses  $\delta \circ (\delta \triangleleft \mathfrak{s})$  or  $\delta \circ (\mathfrak{s} \triangleleft \delta)$ . 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  $\delta \circ (\delta \triangleleft \mathfrak{s})$  on the left and  $\delta \circ (\mathfrak{s} \triangleleft \delta)$  on the right, we find that they are equal:

$$\text{Left side (} \delta \circ (\delta \triangleleft \mathfrak{s}) \text{)} = \text{Right side (} \delta \circ (\mathfrak{s} \triangleleft \delta) \text{)} \tag{6.8}$$

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  $s_0 \rightarrow s_1$  and  $s_1 \rightarrow s_2$  to obtain  $s_0 \rightarrow s_2$ , before running that together with  $s_2 \rightarrow s_3$  to obtain  $s_0 \rightarrow s_3$ , as we see in the upper left box. Meanwhile, on the right, we run together  $s_1 \rightarrow s_2$  and  $s_2 \rightarrow s_3$  to obtain  $s_1 \rightarrow s_3$ , before running  $s_0 \rightarrow s_1$  together with our newly obtained  $s_1 \rightarrow s_3$  to again obtain  $s_0 \rightarrow s_3$  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  $\delta$ , it also makes intuitive sense.

*Exercise 6.9.* Using only the contents of the blue boxes and the target and run functions, write down the equation that we can read off of (6.8) expressing the associativity of the “running together” operation.  $\diamond$

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  $\delta$  is the identity, which we already knew from Section 6.1.3; 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

$$\begin{aligned} \text{tgt}(s_0, \text{run}(s_0, s_0 \rightarrow s_1, s_1 \rightarrow s_2)) &= \text{tgt}(s_0, s_0 \rightarrow s_2) \\ &= s_2 \\ &= \text{tgt}(s_1, s_1 \rightarrow s_2) \\ &= \text{tgt}(\text{tgt}(s_0, s_0 \rightarrow s_1), s_1 \rightarrow s_2), \end{aligned}$$

which is exactly what we wanted in (6.5). In English, this says that when we run together  $s_0 \rightarrow s_1$  and  $s_1 \rightarrow s_2$ , the new direction’s target is the same as the direction of  $s_1 \rightarrow s_2$ , 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 (6.5) are captured by the equality of lenses  $\delta \circ (\delta \triangleleft s) = \delta \circ (s \triangleleft \delta)$ . Equivalently, the following diagram in **Poly** commutes:

$$\begin{array}{ccc} s & \xrightarrow{\delta} & s \triangleleft s \\ \delta \downarrow & & \downarrow s \triangleleft \delta \\ s \triangleleft s & \xrightarrow{\delta \triangleleft s} & s \triangleleft s \triangleleft s. \end{array} \quad (6.10)$$

Another way to say this is that  $\delta$  is *coassociative*: while  $\delta$  is only a lens  $s \rightarrow s^{\triangleleft 2}$  as defined, the commutativity of (6.10) tells us that the two ways of getting a lens  $s \rightarrow s^{\triangleleft 3}$  out of  $\delta$  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  $\delta$  induces a canonical lens  $s \rightarrow s^{\triangleleft 3}$ , which we will call  $\delta^{(3)}$ , as it has 3 copies of  $s$  in its codomain. Armed with this new lens, we can model three steps through a system  $\varphi: s \rightarrow p$  with interface  $p \in \mathbf{Poly}$  as the composite lens

$$s \xrightarrow{\delta^{(3)}} s^{\triangleleft 3} \xrightarrow{\varphi^{\triangleleft 3}} p^{\triangleleft 3}.$$

In fact, coassociativity guarantees that  $\delta$  induces a canonical lens  $\delta^{(n)}: s \rightarrow s^{\triangleleft n}$  for every integer  $n \geq 2$ , starting with  $\delta^{(2)} := \delta$ .<sup>1</sup> For concreteness, we could then define

<sup>1</sup>Perhaps this notation seems a little unnatural, but it helps to think of the original  $\delta: s \rightarrow s \triangleleft s$  as the  $n = 2$  case of a generalized transition lens modeling  $n$  steps through the state system.

$\delta^{(n)}$  for  $n > 2$  inductively by  $\delta^{(n)} := \delta \circ (\delta^{(n-1)} \triangleleft s)$ , or just as well by  $\delta^{(n)} := \delta \circ (s \triangleleft \delta^{(n-1)})$  or even  $\delta^{(n)} := \delta \circ (\delta^{(\ell)} \triangleleft \delta^{(m)})$  for some pair of integers  $\ell, m > 1$  satisfying  $\ell + m = n$ . Regardless, the coassociativity of  $\delta$  means that it doesn't matter how we build a lens  $s \rightarrow s^{\triangleleft(n+1)}$  out of  $\delta, \circ, \triangleleft$ , and identity lenses: we'll always end up with the same lens. We will state this in more generality in Proposition 6.20, but here's some practice with the  $n = 4$  case for a taste of what's to come.

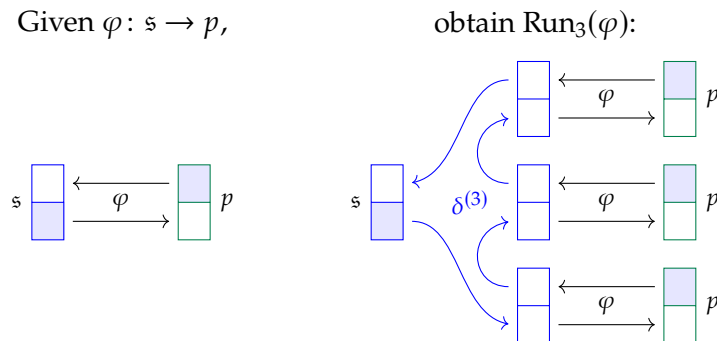
*Exercise 6.11.* 1. Say we know nothing about  $s$  or  $\delta$  apart from the fact that  $s \in \mathbf{Poly}$  and that  $\delta$  is a lens  $s \rightarrow s \triangleleft s$ . List all the ways to obtain a lens  $s \rightarrow s^{\triangleleft 4}$  using only copies of  $\delta, \text{id}_s, \triangleleft$ , and  $\circ$ . (You may write  $s$  for  $\text{id}_s$ .)  
 2. Now assume that (6.10) commutes. Show that all the lenses on your list are equal. (Hint: Use the fact that  $(f \circ g) \triangleleft (h \circ k) = (f \triangleleft h) \circ (g \triangleleft k)$  for lenses  $f, g, h, k$ ).  $\diamond$

### 6.1.5 Running dynamical systems

Finally, we are ready to fulfill our promise from way back in Example 3.43 by using the language of  $\mathbf{Poly}$  to describe stepping through a dynamical system  $n$  times for arbitrary  $n \in \mathbb{N}$ . Given a dynamical system  $\varphi: s \rightarrow p$  with interface  $p \in \mathbf{Poly}$ , we can construct a new dynamical system that we call  $\text{Run}_n(\varphi)$ , with the same state system  $s$  but a new interface  $p^{\triangleleft n}$ , by defining  $\text{Run}_n(\varphi) := \delta^{(n)} \circ \varphi^{\triangleleft n}$ . Visually, we define  $\text{Run}_n(\varphi)$  so that the following diagram commutes:

$$\begin{array}{ccccc} s & \xrightarrow{\delta^{(n)}} & s^{\triangleleft n} & \xrightarrow{\varphi^{\triangleleft n}} & p^{\triangleleft n} \\ & \searrow & & \nearrow & \\ & & \text{Run}_n(\varphi) & & \end{array}$$

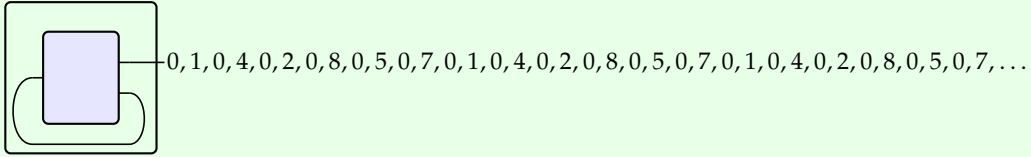
One way to think of this is that  $\text{Run}_n(\varphi)$  is a sped-up version of  $\varphi$ : one step through  $\text{Run}_n(\varphi)$  is equivalent to  $n$  steps through  $\varphi$ . But this is just because a single interaction with the interface  $p^{\triangleleft n}$  models a sequence of  $n$  interactions with the interface  $p$ , as detailed in Section 5.1.4 and Example 5.49. So  $\text{Run}_n(\varphi)$  repackages  $n$  cycles through  $\varphi$  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  $\delta$  does this for the  $n = 2$  case in Example 5.50, and here's a polybox picture for the  $n = 3$  case:



Notice that we have only defined  $\delta^{(n)}$ , and thus  $\text{Run}_n(\varphi)$ , for integers  $n \geq 2$ . But  $n = 0$  runs through  $n$  is doing nothing, modeled by the do-nothing enclosure  $\epsilon: \mathfrak{s} \rightarrow y$ , while  $n = 1$  run through  $\varphi$  is modeled by  $\varphi: \mathfrak{s} \rightarrow \mathfrak{s}$  itself. So we want  $\text{Run}_0(\varphi) = \epsilon$  and  $\text{Run}_1(\varphi) = \varphi$ ; we can achieve this by setting  $\delta^{(0)} := \epsilon$  and  $\delta^{(1)} := \text{id}_{\mathfrak{s}}$ . Here we should think of the do-nothing enclosure  $\delta^{(0)}$  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 6.12.* Verify that when  $\delta^{(0)} = \epsilon$  and  $\delta^{(1)} = \text{id}_{\mathfrak{s}}$ , if  $\text{Run}_n(\varphi)$  is defined as  $\delta^{(n)} \circ \varphi^{\ast n}$  for all  $n \in \mathbb{N}$ , then  $\text{Run}_0(\varphi) = \epsilon$  and  $\text{Run}_1(\varphi) = \text{id}_{\mathfrak{s}}$ .  $\diamond$

*Example 6.13* (Returning every other output). In Exercise 3.67, we built a dynamical system  $\varphi: Sy^S \rightarrow \mathbb{N}y$  that outputs 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  $Sy^S \rightarrow \mathbb{N}y$  that acts like  $\varphi$ , except that it only returns every other output that  $\varphi$  returns.

We could build such a system from scratch—or we can simply start from  $\varphi$  and apply  $\text{Run}_2$ , yielding a system  $\text{Run}_2(\varphi): Sy^S \rightarrow \mathbb{N}y \triangleleft \mathbb{N}y \cong \mathbb{N}^2y$  that returns the outputs of  $\varphi$  two at a time:

$$(0, 1), (0, 4), (0, 2), (0, 8), (0, 5), (0, 7), (0, 1), (0, 4), (0, 2), (0, 8), (0, 5), (0, 7), \dots$$

Then we just need to compose  $\text{Run}_2(\varphi)$  with a lens  $\pi_2: \mathbb{N}^2y \rightarrow \mathbb{N}y$  equal to the second coordinate projection on positions (and the identity on directions) to extract the outputs we want. The new system  $Sy^S \rightarrow \mathbb{N}y$  that skips over every output of  $\varphi$  is therefore the following composite:

$$\begin{array}{c}
 Sy^S \xrightarrow{\delta^{(2)}} Sy^S \triangleleft Sy^S \xrightarrow{\varphi^{\ast 2}} \mathbb{N}y \triangleleft \mathbb{N}y \cong \mathbb{N}^2y \xrightarrow{\pi_2} \mathbb{N}y. \\
 \searrow \text{Run}_2(\varphi) \nearrow
 \end{array}$$

We can apply this technique in general to skip (or otherwise act on) the outputs of a dynamical system at regular intervals.

One drawback of the  $\text{Run}_n(-)$  operation is that we need to keep track of a separate morphism  $Sy^S \rightarrow p^{\ast n}$  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 7; but for now, let us investigate what's really going on with our state systems algebraically.

### 6.1.6 State systems as comonoids

It turns out that objects equipped with morphisms like those in Sections 6.1.1 and 6.1.2 that satisfy the commutative diagrams from Sections 6.1.3 and 6.1.4 are well-known to category theorists.

**Definition 6.14** (Comonoid). In a monoidal category  $(\mathbf{C}, y, \triangleleft)$ , a *comonoid*  $\mathcal{C} := (c, \epsilon, \delta)$  consists of

- an object  $c \in \mathbf{C}$ , called the *carrier*;
- a morphism  $\epsilon: c \rightarrow y$  in  $\mathbf{C}$ , called the *eraser* (or the *counit*); and
- a morphism  $\delta: c \rightarrow c \triangleleft c$  in  $\mathbf{C}$ , called the *duplicator* (or the *comultiplication*);

such that the following diagrams, collectively known as the *comonoid laws*, commute:

$$\begin{array}{ccccc}
 y \triangleleft c & \xlongequal{\quad} & c & \xlongequal{\quad} & c \triangleleft y \\
 \nwarrow \epsilon \triangleleft c & & \downarrow \delta & & \nearrow c \triangleleft \epsilon \\
 & & c \triangleleft c, & & 
 \end{array} \tag{6.15}$$

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

$$\begin{array}{ccc}
 c & \xrightarrow{\delta} & c \triangleleft c \\
 \delta \downarrow & & \downarrow c \triangleleft \delta \\
 c \triangleleft c & \xrightarrow{\delta \triangleleft c} & c \triangleleft c \triangleleft c,
 \end{array} \tag{6.16}$$

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  $\mathcal{C}$  in  $(\mathbf{Poly}, y, \triangleleft)$  as a *polynomial comonoid*.

*Remark 6.17.* 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 6.40.) 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  $\epsilon$  and  $\delta$  that we will so often use to label them. The word “duplicator” comes from the fact that  $\delta: c \rightarrow c \triangleleft c$  effectively turns one  $c$  into two, while the “eraser”  $\epsilon: c \rightarrow y$  erases the  $c$  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 6.18.* 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 as arenas than as functors, we’ll favor the term comonoid over comonad.

*Example 6.19* (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 enclosure  
and whose duplicator is the transition lens.*

The comonoid structure on a state system  $s$  is what allows us to write canonical lenses  $s \rightarrow s^{\triangleleft n}$  for any  $n \in \mathbb{N}$ . We can then model  $n$  steps through a dynamical system  $\varphi: s \rightarrow p$  with interface  $p \in \mathbf{Poly}$  by composing this canonical lens with  $\varphi^{\triangleleft n}$  to obtain a “sped-up” dynamical system  $\text{Run}_n(\varphi)$ . This new system has the same state system  $s$ , but its interface is now  $p^{\triangleleft n}$ .

The canonicity of  $s \rightarrow s^{\triangleleft n}$  is due to the following standard result about comonoids, which can be proved inductively.

**Proposition 6.20** (Defining  $\delta^{(n)}$ ). Given a comonoid  $(c, \epsilon, \delta)$ , let  $\delta^{(n)}: c \rightarrow c^{\triangleleft n}$  be given as follows. Let  $\delta^{(0)} := \epsilon$  and inductively define  $\delta^{(n+1)} := \delta \circ (\delta^{(n)} \triangleleft c)$  for all  $n \in \mathbb{N}$ . Then we have the following:

- (a)  $\delta^{(n)}$  is a morphism  $c \rightarrow c^{\triangleleft n}$  for all  $n \in \mathbb{N}$ ;
- (b)  $\delta^{(1)} = c = \text{id}_c$ ;
- (c)  $\delta^{(2)} = \delta$ ; and
- (d)  $\delta^{(n)} = \delta \circ (\delta^{(k)} \triangleleft \delta^{(n-k)})$  for all  $k, n \in \mathbb{N}$  with  $k \leq n$ , so our choice of morphism  $c \rightarrow c^{\triangleleft(n+1)}$  is canonical.

*Proof.* We leave parts (a), (b), and (c) for Exercise 6.21. Part (d) amounts to coassociativity.  $\square$

We’ll continue to use the notation introduced here throughout for general comonoids.



*Exercise 6.21.* Prove the first three parts of Proposition 6.20.

1. Prove part (a).
2. Prove part (b).
3. Prove part (c).

◇

*Example 6.22* (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?

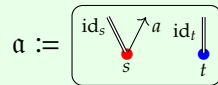
The answer turns out to be no. After all, there is one fact about state systems from Section 6.1.2 that we did not encode in **Poly**: for a fixed state  $s \in \mathfrak{s}(1)$ , the target function  $\mathfrak{s}[s] \rightarrow \mathfrak{s}(1)$  assigning directions at  $s$  to their target states is a bijection.

Nothing in our comonoid laws guarantees this bijectivity. An arbitrary polynomial comonoid might assign 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 assign *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 6.23.) 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 6.23* (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  $\alpha := \{s\}y^{\{\text{id}_s, a\}} + \{t\}y^{\{\text{id}_t\}} \cong y^2 + y$ . Here is its tree picture:

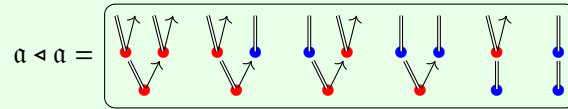


Notice that we have drawn one direction out of each position— $\text{id}_s$  and  $\text{id}_t$ —with a double bar. We let these be the directions that the eraser  $\epsilon: \alpha \rightarrow y$  picks out. The

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  $\epsilon$  makes like so:



Now we need a duplicator  $\delta: a \rightarrow a \triangleleft a$ . Before we define it, let's draw out  $a \triangleleft a$  to see what it looks like. Remember that we need to glue corollas of  $a$  to leaves of  $a$  in every possible way:



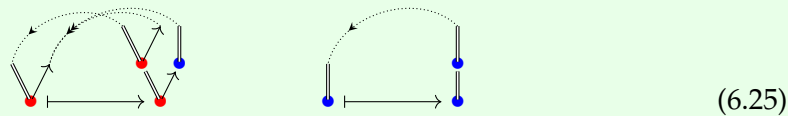
Each of these trees gives a way to match directions out of one position to positions they could lead to. On positions,  $\delta$  will decide which matchings to pick by sending the red  $s$  to one of the four positions on the left and the blue  $t$  to one of the two positions on the right. We want 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,  $\delta$  looks like this:



As in Example 6.6, we can interpret this as telling us how to “bend” the arrows of  $a$  so that they point to other positions:

$$\text{Diagram: A red dot with a self-loop arrow pointing to a blue dot with a self-loop arrow.} \quad (6.24)$$

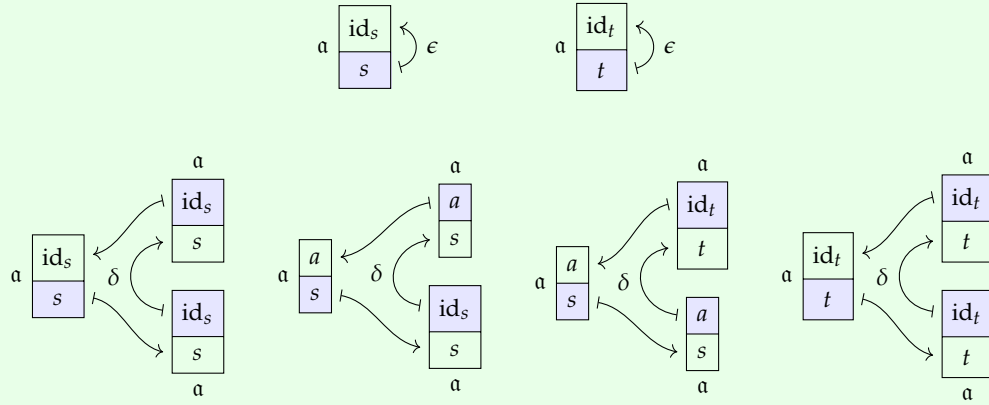
Meanwhile, on directions,  $\delta$  should tell us how to run two directions together into one. Fortunately, there's not much for us to do here—we know that if one of the two directions  $\delta$  runs together is one of the double-barred directions that the eraser picked out, then  $\delta$  should ignore that “do-nothing” direction and yield the other direction (again, the erasure laws ensure this). Here's what that looks like:



(6.25)

And that's all we need to specify the triple  $(\alpha, \epsilon, \delta)$ .

Here are  $\epsilon: \alpha \rightarrow y$  and  $\delta: \alpha \rightarrow \alpha \triangleleft \alpha$  again, in terms of polyboxes.



Of course, we have yet to check that  $(\alpha, \epsilon, \delta)$  really is a comonoid, i.e. that the diagrams in (6.15) and (6.16) commute. We leave that for Exercise 6.26.

*Exercise 6.26.* Verify that  $(\alpha, \epsilon, \delta)$  as defined in Example 6.23 obeys the erasure laws in (6.15) and the coassociative law in (6.16).  $\diamond$

*Exercise 6.27.* Show that if  $B$  is a set, then there exists a unique comonoid structure on the linear polynomial  $By$ .  $\diamond$

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 (or simply remembered what we told you back in Proposition 1.20): 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.

## 6.2 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 6.28** (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!*

### 6.2.1 Translating between polynomial comonoids and categories

First, we describe how to translate between the carrier  $\mathfrak{c}$  of a comonoid  $\mathcal{C} := (\mathfrak{c}, \delta, \epsilon)$  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  $\mathcal{C}$ :

$$\mathfrak{c}(1) = \text{Ob } \mathcal{C}. \quad (6.29)$$

Then for each such position or object  $i$ , the  $\mathfrak{c}[i]$ -directions are the morphisms of  $\mathcal{C}$  with domain  $i$ :

$$\mathfrak{c}[i] = \sum_{j \in \text{Ob } \mathcal{C}} \mathcal{C}(i, j). \quad (6.30)$$

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.<sup>2</sup> 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 6.31** (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<sup>3</sup>

$$\mathcal{C}[i] := \sum_{j \in \text{Ob } \mathcal{C}} \mathcal{C}(i, j).$$

Then the *polynomial carrier*, or simply *carrier*, of  $\mathcal{C}$  is the polynomial

$$\sum_{i \in \text{Ob } \mathcal{C}} y^{\mathcal{C}[i]}.$$

<sup>2</sup>Except, perhaps, in the context of coslice categories.

So everything we have said so far about the correspondence from Theorem 6.28 can be summarized by saying that it preserves carriers: the carrier of the category  $\mathcal{C}$  is the carrier  $\mathfrak{c}$  of the comonoid  $\mathcal{C}$ , so that  $\text{Ob } \mathcal{C} = \mathfrak{c}(1)$  and  $\mathcal{C}[i] = \mathfrak{c}[i]$ .

*Remark 6.32.* 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 (6.29) and (6.30) really can be just strict equalities. This is why we are comfortable naming a “one-to-one correspondence” in Theorem 6.28 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 6.33.* 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

$$\boxed{0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow \cdots}$$

where there is a unique morphism  $m \rightarrow n$  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).  $\diamond$

But a category  $\mathcal{C}$  is more than its carrier polynomial, just as a comonoid  $\mathcal{C}$  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

---

<sup>3</sup>We may also write  $f: i \rightarrow \_$  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  $\epsilon$  or the duplicator  $\delta$  of  $\mathcal{C}$ , 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.

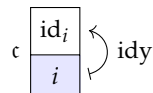
Comonoid	$\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$	Category	$\mathcal{C}$
carrier	$i \in \mathfrak{c}(1)$	objects	$i \in \text{Ob } \mathcal{C}$
	$f \in \mathfrak{c}[i]$	morphisms	$f: i \rightarrow \_$
eraser	$\epsilon_1: \mathfrak{c}(1) \rightarrow 1$	—	—
	$\epsilon_i^\# : 1 \rightarrow \mathfrak{c}[i]$	identities	$\text{id}_i: i \rightarrow \_$
duplicator	$\delta_1: \mathfrak{c}(1) \rightarrow (\mathfrak{c} \triangleleft \mathfrak{c})(1)$	codomains*	$\text{cod}: \mathcal{C}[i] \rightarrow \text{Ob } \mathcal{C}$
	$\delta_i^\# : (\mathfrak{c} \triangleleft \mathfrak{c})[\delta_1(i)] \rightarrow \mathfrak{c}[i]$	composition*	$\circ$
right erasure law		* right identity law	
left erasure law		codomains of identities	$\text{cod } \text{id}_i = i$
		left identity law	
coassociative law		codomains of composites	$\text{cod}(f \circ g) = \text{cod } g$
		associative law of composition	

Note that the on-positions function of  $\epsilon$ , 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 6.1, so we'll go through it a little faster this time for the general case.

### The eraser assigns identities

We know that the eraser  $\epsilon: \mathfrak{c} \rightarrow y$  can be identified with a dependent function  $(i \in \mathfrak{c}(1)) \rightarrow [i]$ , sending each position  $i \in (1)$  to a  $[i]$ -direction. In terms of our category  $\mathcal{C}$ , the eraser sends each object  $i \in \text{Ob } \mathcal{C}$  to a morphism  $i \rightarrow \_$ . 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  $\epsilon$  like so:

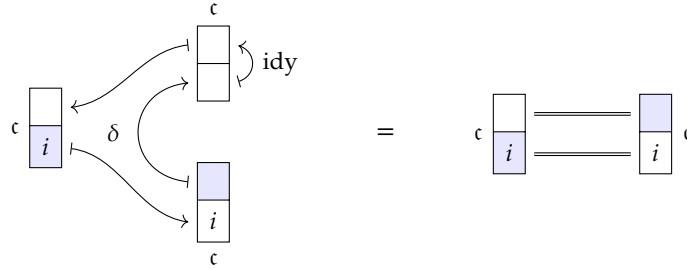


Here we have given the label  $\text{id}_c$  to the arrow sending objects to their identity morphisms.

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

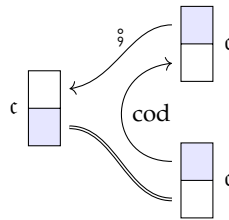
Keeping our label  $\text{id}_c$  for the arrow in  $\epsilon$ , the right erasure law  $\delta \circ (\epsilon \triangleleft \epsilon) = \text{id}_c$  from (6.15) can be drawn in polyboxes like so:



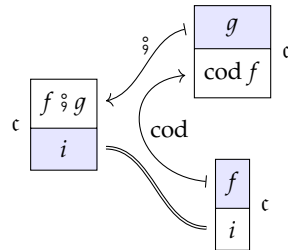
We have only filled in a few of the boxes, but that is enough to interpret what the right erasure law tells us on positions: that the bottom arrow of the duplicator must be the identity function on  $c(1)$ . Equipped with this knowledge, we can focus our attention on the other two arrows of  $\delta$ .

### The duplicator assigns codomains and composites

In fact, in the polybox picture for  $\delta: c \rightarrow c \triangleleft c$ , the middle arrow specifies codomains, and the top arrow specifies composition. We therefore label these arrows as follows:<sup>4</sup>



To check that this makes sense, we fill in the boxes:

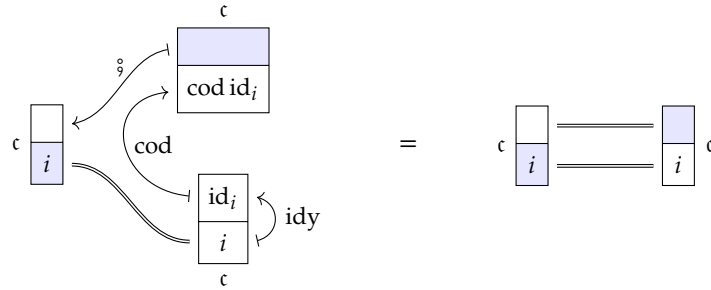


<sup>4</sup>Compare these labels to the names “target” and “run” that we gave to the arrows of a state system’s transition lens.

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  $\delta$  takes an object  $i \in \text{Ob } \mathcal{C}$  and a morphism  $f: i \rightarrow \_$  in  $\mathcal{C}$  and assigns another object  $\text{cod } f \in \text{Ob } \mathcal{C}$  to be the codomain of  $f$ . It then takes another morphism  $g: \text{cod } f \rightarrow \_$  in  $\mathcal{C}$  and assigns a morphism  $f \circ g: i \rightarrow \_$  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  $f \circ g$  matches the domain of  $f$ , as it should.

### The left erasure law on positions: codomains of identities

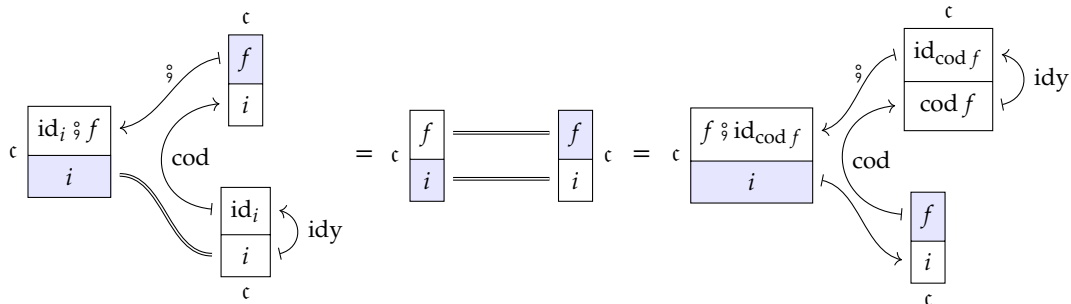
As with the right erasure law, we can partially fill in the polyboxes for the left erasure law  $\delta \circ (\epsilon \triangleleft c) = \text{id}_c$  from (6.15) to read what it says on positions:



So the left erasure law on positions guarantees that  $\text{cod } \text{id}_i = i$  for all  $i \in \text{Ob } \mathcal{C}$ . 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  $\text{cod } \text{id}_i$  on the left, which we now know we can do), while the right equality depicts the right erasure law:





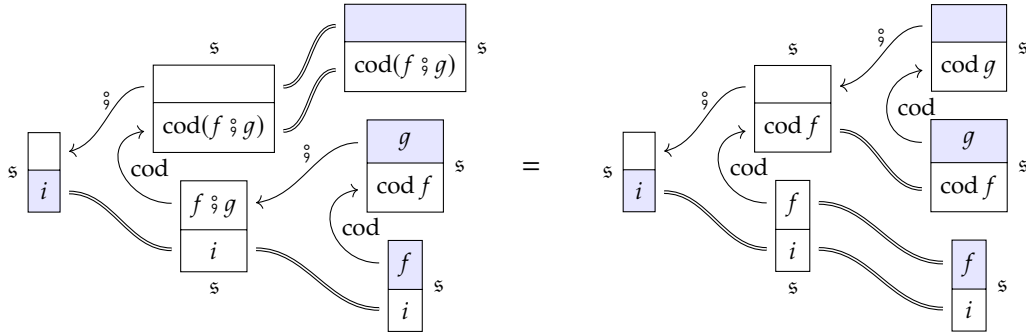
We find that on directions, the erasure laws state that for every object  $i \in \text{Ob } \mathcal{C}$  and morphism  $f: i \rightarrow \_$  in  $\mathcal{C}$ ,

$$\text{id}_i \circ f = f = f \circ \text{id}_{\text{cod } f}.$$

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 (6.16),  $\delta \circ (\delta \triangleleft s) = \delta \circ (s \triangleleft \delta)$ . 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:



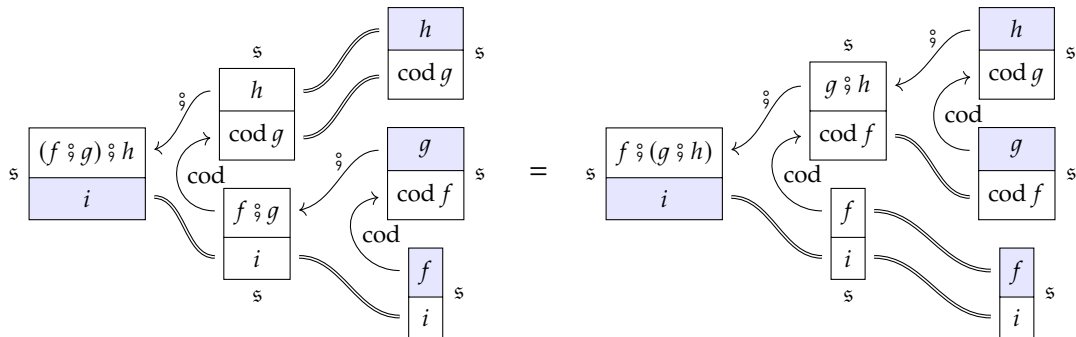
So on positions, the coassociative law states that given an object  $i \in \text{Ob } \mathcal{C}$  and morphisms  $f: i \rightarrow \_$  and  $g: \text{cod } f \rightarrow \_$  in  $\mathcal{C}$ ,

$$\text{cod}(f \circ g) = \text{cod } g.$$

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  $\text{cod } g$  for  $\text{cod}(f \circ g)$  on the left, which we now know we can do):



Thus, on directions, the coassociative law states that given an object  $i \in \text{Ob } \mathcal{C}$  and morphisms  $f: i \rightarrow \_$ ,  $g: \text{cod } f \rightarrow \_$ , and  $h: \text{cod } g \rightarrow \_$  in  $\mathcal{C}$ ,

$$(f \circ g) \circ h = f \circ (g \circ h).$$

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 6.28.

### 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  $\delta^{(3)}: c \rightarrow c^{\triangleleft 3}$ , as defined in Proposition 6.20. 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  $i \xrightarrow{f \circ g \circ h} \_$ , and (co)associativity ensures this is well-defined.

In general,  $\delta^{(n)}: c \rightarrow c^{\triangleleft n}$  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  $\delta^{(1)} = \text{id}_c$  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  $c^{\triangleleft n}$  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.

## 6.2.2 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  $f: c \rightarrow d$  is the *only* morphism  $c \rightarrow d$ .<sup>5</sup> Composition in preorders is easy to describe, because the composite of  $c \rightarrow d$  and  $d \rightarrow e$  is always just the unique arrow  $c \rightarrow e$ . As such, preorders are some of the simplest examples of categories to consider—we already saw several in Exercise 6.33—so let us interpret these as comonoids first.

*Example 6.34.* Let us revisit Example 6.23, where we first wrote down a comonoid that was not a state system. We defined  $\alpha := \{s\}y^{\{\text{id}_s, a\}} + \{t\}y^{\{\text{id}_t\}} \cong y^2 + y$  and gave it a comonoid structure, with eraser  $\epsilon: \alpha \rightarrow y$  specifying directions  $\text{id}_s$  and  $\text{id}_t$  and duplicator  $\delta: \alpha \rightarrow \alpha \triangleleft \alpha$  pointing the direction  $a$  at  $t$ .

Looking at the picture we drew of the comonoid in (6.24), it should come as no surprise that the corresponding category  $\mathcal{A}$  is the *walking arrow category*, which is a

<sup>5</sup>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  $\alpha$ -positions  $s$  and  $t$ . It has two morphisms with domain  $s$ , the  $\alpha[s]$ -directions  $\text{id}_s$  and  $a$ ; and one morphism with domain  $t$ , the  $\alpha[t]$ -direction  $\text{id}_t$ . The morphisms  $\text{id}_s$  and  $\text{id}_t$  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 6.35.* 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  $\epsilon$ .
3. Characterize the duplicator  $\delta$ .

◇

*Exercise 6.36.* We showed in Exercise 6.27 that for any set  $B$ , the linear polynomial  $By$  has a unique comonoid structure. To what category does this comonoid correspond?

◇

*Exercise 6.37.* 1. Find a comonoid structure for the polynomial  $p := y^{n+1} + ny$  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”?

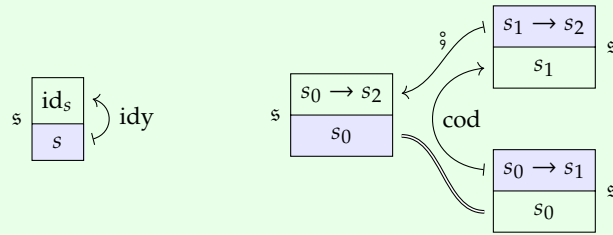
◇

*Example 6.38* (State systems as categories). We know that every state system  $\mathfrak{s} \cong Sy^{\mathfrak{s}}$  with its do-nothing enclosure  $\epsilon: \mathfrak{s} \rightarrow y$  and its transition lens  $\delta: \mathfrak{s} \rightarrow \mathfrak{s} \triangleleft y$  is a comonoid, so what category  $\mathcal{S}$  does  $(\mathfrak{s}, \epsilon, \delta)$  correspond to?

Recall from Example 6.22 that state systems are exactly those comonoids whose codomain (i.e. “target”) functions  $\text{cod}: \mathfrak{s}[s] \rightarrow \mathfrak{s}(1)$  for  $s \in \mathfrak{s}(1)$  are bijections. That is, from every object  $s \in \text{Ob } \mathcal{S} = \mathfrak{s}(1)$ , there is exactly 1 morphism to every object  $t \in \text{Ob } \mathcal{S}$ . 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 enclosure of  $\mathfrak{s}$  from (6.1) and the

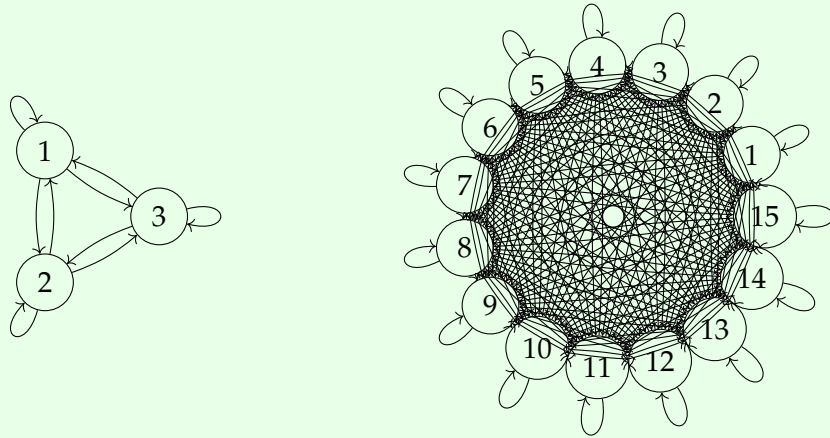
transition lens of  $\mathfrak{s}$  from (6.4), this time with our new arrow labels, as a sanity check:



Indeed, we had already been writing the directions of  $\mathfrak{s}$  as arrows  $s \rightarrow t$ , knowing that each was uniquely specified by its source  $s$  and its target  $t$  in  $\mathfrak{s}(1)$ . And in Section 6.1.3, we had already noted that  $\text{id}_s$  was just the arrow  $s \rightarrow s$ . 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  $s \rightarrow t$  is an isomorphism with inverse  $t \rightarrow s$ , for these compose as  $\text{id}_s: s \rightarrow s$  in one direction and  $\text{id}_t: t \rightarrow t$  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 3 and on 15, with all maps (even identities) drawn:



The picture on the left should look familiar: it's what we drew in Example 6.6 when took the corolla picture for  $3y^3$  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 6.39.* Let  $S$  be a set. Is there any comonoid structure on  $Sy^S$  other than that of the state category?  $\diamond$

Not only does Example 6.38 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  $\delta: c \rightarrow c \triangleleft c$  of a comonoid takes the corolla picture of  $c$  and "bends the arrows" so that they point to other roots, yielding the underlying graph of a category. Then  $\delta$  on directions collapses two-arrow paths in the graph down to individual arrows, while the eraser  $\epsilon: c \rightarrow y$  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 arenas and 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 arena 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  $*$ :  $M \times M \rightarrow M$  is the binary operation.

*Example 6.40* (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  $\epsilon: y^M \rightarrow y$  picks out the identity morphism  $e \in M$  on directions, while the duplicator  $\delta: y^M \rightarrow y^M \triangleleft y^M \cong y^{M \times M}$  can be identified with the binary

operation  $*$ :  $M \times M \rightarrow M$ . (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  $(y^M, \epsilon, \delta)$  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 6.41.* Verify Example 6.40 by showing that  $(M, e, *)$  satisfies the unitality and associativity requirements of a monoid in  $(\mathbf{Set}, 1, \times)$  if and only if  $(y^M, \epsilon, \delta)$  satisfies the erasure and coassociativity requirements of a comonoid in  $(\mathbf{Poly}, y, \triangleleft)$ .  $\diamond$

*Example 6.42* (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,  $\epsilon: y^{\mathbb{Z}/n\mathbb{Z}} \rightarrow y$  picks out the “current” element via its  $X$ -component  $\epsilon \triangleleft X: y^{\mathbb{Z}/n\mathbb{Z}} \triangleleft X \rightarrow y \triangleleft X$ , which is just a function  $\epsilon_X: X^{\mathbb{Z}/n\mathbb{Z}} \rightarrow X$ ; and  $\delta$  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 6.43* (Monoid actions). Suppose that  $(M, e, *)$  is a monoid,  $S$  is a set, and  $\alpha: S \times M \rightarrow S$  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

$$\begin{array}{ccc} S \times 1 & \xrightarrow{S \times e} & S \times M \\ & \searrow & \downarrow \alpha \\ & & S \end{array} \quad \text{and} \quad \begin{array}{ccc} S \times M \times M & \xrightarrow{S \times *} & S \times M \\ \downarrow \alpha \times M & & \downarrow \alpha \\ S \times M & \xrightarrow{\alpha} & S \end{array}$$

commute.

Then there is an associated category  $M\mathcal{A}$  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  $(Sy^M, \epsilon, \delta)$ , as we will see in the next exercise.

*Exercise 6.44.* With notation as in Example 6.43, characterize the comonoid structure on  $Sy^M$ .

1. How can we define the erasure  $\epsilon$ ?

2. How can we define the duplicator  $\delta$ ?
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 6.38?  $\diamond$

*Example 6.45* (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  $\bar{b} \in B^{\mathbb{N}}$  as

$$\bar{b} := (b_0 \rightarrow b_1 \rightarrow b_2 \rightarrow b_3 \rightarrow \cdots),$$

with  $b_n \in B$  for each  $n \in \mathbb{N}$ .

Then there is a monoid action  $\tau: B^{\mathbb{N}} \times \mathbb{N} \rightarrow B^{\mathbb{N}}$  for which

$$\tau(\bar{b}, n) := (b_n \rightarrow b_{n+1} \rightarrow b_{n+2} \rightarrow b_{n+3} \rightarrow \cdots).$$

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  $\tau(\bar{b}, 0) = \bar{b}$  and that  $\tau(\bar{b}, m + n) = \tau(\tau(\bar{b}, m), n)$ .

So by Example 6.43, the corresponding comonoid is carried by  $B^{\mathbb{N}}y^{\mathbb{N}}$ . Each  $B$ -stream  $\bar{b}$  is a position, and each  $n \in \mathbb{N}$  is a direction at  $\bar{b}$  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  $\bar{b}$  starting at  $b_n$ . Indeed, this  $B$ -stream is exactly  $\tau(\bar{b}, n)$ , the codomain assigned to the direction  $n$  at  $\bar{b}$ .

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  $\bar{b} \rightarrow \bar{b}'$  consist of every way in which  $\bar{b}'$  can be viewed as a contiguous substream of  $\bar{b}$ : that is, there is a morphism  $n: \bar{b} \rightarrow \bar{b}'$  for each  $n \in \mathbb{N}$  satisfying

$$(b_n \rightarrow b_{n+1} \rightarrow \cdots) = (b'_0 \rightarrow b'_1 \rightarrow \cdots).$$

The identity on  $\bar{b}$  is given by  $0: \bar{b} \rightarrow \bar{b}$ ; and the composite of two morphisms is the sum of the corresponding natural numbers, as a substream of a substream of  $\bar{b}$  is just a substream of  $\bar{b}$  shifted by the appropriate amount.

We will see this category again in Example 7.38.

*Exercise 6.46.* 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 6.47** (Degree, linear). Let  $\mathcal{C}$  be a category and  $c \in \text{Ob } \mathcal{C}$  an object. The *degree* of  $c$ , denoted  $\deg(c)$ , is the set of arrows in  $\mathcal{C}$  that emanate from  $c$ .

If  $\deg(c) \cong 1$ , we say that  $c$  is *linear*. If  $\deg(c) \cong n$  for  $n \in \mathbb{N}$ , we say  $c$  has *degree*  $n$ .

- Exercise 6.48.*
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?

◇

## 6.3 Morphisms of polynomial comonoids are cofunctors

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 *cofunctor*.

### 6.3.1 Introducing comonoid morphisms and cofunctors

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.

**Definition 6.49** (Comonoid morphism). Given a monoidal category  $(\mathcal{C}, y, \triangleleft)$  with comonoids  $\mathcal{C} := (c, \epsilon, \delta)$  and  $\mathcal{C}' := (c', \epsilon', \delta')$ , a *comonoid morphism* (or *morphism of comonoids*)  $\mathcal{C} \rightarrow \mathcal{C}'$  is a morphism  $F: c \rightarrow c'$  in  $\mathcal{C}$  for which the following diagrams commute:

$$\begin{array}{ccc}
 c & \xrightarrow{F} & c' \\
 \epsilon \downarrow & & \downarrow \epsilon' \\
 y & \xlongequal{\quad} & y
 \end{array} \tag{6.50}$$



called the *eraser preservation law* (we say  $F$  *preserves erasure*); and

$$\begin{array}{ccc} c & \xrightarrow{F} & c' \\ \delta \downarrow & & \downarrow \delta' \\ c \triangleleft c & \xrightarrow{F \triangleleft F} & c' \triangleleft c' \end{array} \quad (6.51)$$

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  $\mathbf{Comon}(\mathcal{C})$  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 6.52.* 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 6.49, 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.

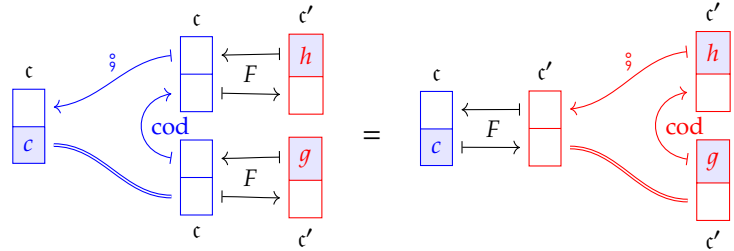
This will show that  $\mathbf{Comon}(\mathcal{C})$  is indeed a subcategory of  $\mathcal{C}$ .  $\diamond$

Notice that we were very careful in how we stated Theorem 6.28: while we asserted the existence of an isomorphism-preserving one-to-one correspondence between the objects of  $\mathbf{Comon}(\mathbf{Poly})$  and  $\mathbf{Cat}$ , 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 *cofunctors*.

How exactly do these maps behave? If we specify Definition 6.49 to the case of  $(\mathbf{Poly}, y, \triangleleft)$ , we can write the eraser preservation law (6.50) in polyboxes as

$$\begin{array}{c} c \\ \boxed{\phantom{c}} \\ c \end{array} \text{idy} = \begin{array}{ccc} c & & c' \\ \boxed{\phantom{c}} & \xleftarrow{\quad} & \boxed{\phantom{c'}} \\ c & \xrightarrow{F} & \end{array} \text{idy} \quad (6.53)$$

and the duplicator preservation law (6.51) in polyboxes as



(6.54)

If we read off the equations from these polyboxes, interpreting polynomial comonoids as categories, we derive the following definition of a cofunctor. (Here (6.56) is equivalent to (6.53), while (6.57) and (6.58) are together equivalent to (6.54).)

**Definition 6.55** (Cofunctor). Let  $\mathcal{C}$  and  $\mathcal{C}'$  be (small) categories. A *cofunctor*  $F: \mathcal{C} \rightarrow \mathcal{C}'$  consists of

- a function  $F: \text{Ob } \mathcal{C} \rightarrow \text{Ob } \mathcal{C}'$  *forward on objects*<sup>6</sup> and
- a function  $F_c^\sharp: \mathcal{C}'[Fc] \rightarrow \mathcal{C}[c]$  *backward on morphisms* for each  $c \in \text{Ob } \mathcal{C}$ ,

satisfying the following conditions, collectively known as the *cofunctor laws*:

- i.  $F$  *preserves identities*:

$$F_c^\sharp \text{id}_{Fc} = \text{id}_c \quad (6.56)$$

for each  $c \in \text{Ob } \mathcal{C}$ ;

- ii.  $F$  *preserves codomains*:

$$F \text{ cod } F_c^\sharp g = \text{cod } g \quad (6.57)$$

for each  $c \in \text{Ob } \mathcal{C}$  and  $g \in \mathcal{C}'[Fc]$ ;

- iii.  $F$  *preserves composites*<sup>7</sup>:

$$F_c^\sharp g \circ F_{\text{cod } F_c^\sharp g}^\sharp h = F_c^\sharp (g \circ h) \quad (6.58)$$

for each  $c \in \text{Ob } \mathcal{C}$ ,  $g \in \mathcal{C}'[Fc]$ , and  $h \in \mathcal{C}'[\text{cod } g]$ .

We let  $\mathbf{Cat}^\sharp \cong \mathbf{Comon}(\mathbf{Poly})$  denote the category of (small) categories and cofunctors.

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 cofunctor includes a lens between its carrier polynomials, cofunctors compose the way lenses do.

<sup>6</sup>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.

<sup>7</sup>In particular, the codomains of either side of (6.58) 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 6.60.

*Exercise 6.59.* Let  $\mathcal{C}, \mathcal{D}, \mathcal{E}$  be categories, and let  $F: \mathcal{C} \rightarrow \mathcal{D}$  and  $G: \mathcal{D} \rightarrow \mathcal{E}$  be cofunctors between them.

1. Characterize the behavior of the identity cofunctor  $\text{id}_{\mathcal{D}}$  on  $\mathcal{D}$ . Where does it send each object? Where does it send each morphism?
2. Characterize the behavior of the composite cofunctor  $F \circ G$ . Where does it send each object and morphism?  $\diamond$

On the surface, functors and cofunctors 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 send morphisms *forward*, while cofunctors send morphisms *backward*. As we work with cofunctors, it will be helpful to remember the following:

*A cofunctor  $F$  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 cofunctors can be, let us note a few more similarities that these two kinds of maps between categories share. For example, cofunctors, like functors, preserve isomorphisms.

**Proposition 6.60.** Let  $F: \mathcal{C} \rightarrow \mathcal{D}$  be a cofunctor,  $c \in \mathcal{C}$  be an object, and  $g: Fc \rightarrow \_$  be an isomorphism in  $\mathcal{D}$ . Then  $F_c^\# g$  is also an isomorphism in  $\mathcal{C}$ .

*Proof.* Let  $c' := \text{cod } F_c^\# g$ , so that  $Fc' = \text{cod } g$  by (6.57), and let  $g^{-1}: Fc' \rightarrow Fc$  be the inverse of  $g$ . Then

$$\text{id}_c = F_c^\# \text{id}_{Fc} \tag{6.56}$$

$$\begin{aligned} &= F_c^\# (g \circ g^{-1}) \\ &= F_c^\# g \circ F_{c'}^\# (g^{-1}), \end{aligned} \tag{6.58}$$

so in particular  $c = \text{cod } \text{id}_c = \text{cod } F_{c'}^\# (g^{-1})$ , and

$$\text{id}_{c'} = F_{c'}^\# \text{id}_{Fc'} \tag{6.56}$$

$$\begin{aligned} &= F_{c'}^\# (g^{-1} \circ g) \\ &= F_{c'}^\# (g^{-1}) \circ F_{\text{cod } F_{c'}^\# (g^{-1})}^\# g \\ &= F_{c'}^\# (g^{-1}) \circ F_c^\# g. \end{aligned} \tag{6.58}$$

Hence  $F_c^\# g$  and  $F_{c'}^\# (g^{-1})$  are inverses, and the result follows.  $\square$

Moreover, isomorphisms in **Cat** correspond to isomorphisms in **Cat**<sup>#</sup>.

*Exercise 6.61.* We’ve justified the “isomorphism-preserving” part of Theorem 6.28 implicitly, but let’s make it explicit.

Recall that two categories  $\mathcal{C}$  and  $\mathcal{D}$  are isomorphic in  $\mathbf{Cat}$  if there exist functors  $F: \mathcal{C} \rightarrow \mathcal{D}$  and  $G: \mathcal{D} \rightarrow \mathcal{C}$  that are mutually inverse, i.e.  $F \circ G$  and  $G \circ F$  are identity functors on  $\mathcal{C}$  and  $\mathcal{D}$ . Similarly,  $\mathcal{C}$  and  $\mathcal{D}$  are isomorphic in  $\mathbf{Cat}^\sharp$  if there exist cofunctors  $H: \mathcal{C} \rightarrow \mathcal{D}$  and  $K: \mathcal{D} \rightarrow \mathcal{C}$  that are mutually inverse, i.e.  $H \circ K$  and  $K \circ H$  are identity cofunctors on  $\mathcal{C}$  and  $\mathcal{D}$ . Show that  $\mathcal{C}$  and  $\mathcal{D}$  are isomorphic in  $\mathbf{Cat}$  if and only if they are isomorphic in  $\mathbf{Cat}^\sharp$ .  $\diamond$

But while isomorphisms in  $\mathbf{Cat}^\sharp$  are the same as those in  $\mathbf{Cat}$ , the non-isomorphisms can be very different. We’ll see this in the examples to come.

### 6.3.2 Examples of cofunctors

From a realm where functors reign supreme, the back-and-forth behavior of cofunctors can seem foreign and counterintuitive. Whereas a functor  $\mathcal{C} \rightarrow \mathcal{D}$  can be thought of as a *diagram*—a picture in the shape of  $\mathcal{C}$ , drawn with the objects and arrows of  $\mathcal{D}$ —cofunctors are much more like the dynamical systems of Chapter 3.<sup>8</sup>

That is, a cofunctor  $F: \mathcal{C} \rightarrow \mathcal{D}$  is a way of interacting with the states (objects) and transitions (morphisms) within  $\mathcal{C}$  by way of  $\mathcal{D}$ . Imagine the cofunctor 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  $\text{cod } g$  on the outside,  $F_c^\sharp g$  is followed from  $c$  to  $\text{cod } F_c^\sharp g$  on the inside. But codomain preservation guarantees that the new state  $\text{cod } g$  on the outside is equal to what the box would want to display in lieu of the new state  $\text{cod } F_c^\sharp g$  on the inside, as  $\text{cod } g = F \text{ cod } F_c^\sharp g$ . Then the process repeats in a manner compatible with identities and composition.

Here we give a variety of examples of cofunctors 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  $c$ .

#### Cofunctors to preorders

Given a cofunctor 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.

<sup>8</sup>In fact, we will see in Chapter 7 that cofunctors generalize our dynamical systems.

*Example 6.62* (Cofunctors to discrete categories). The discrete category on a set  $S$  is the category with objects in  $S$  and only identity morphisms; its carrier is  $Sy$ . So a cofunctor  $F: \mathcal{C} \rightarrow Sy$  is completely determined by its behavior on objects: to preserve identities, it can only send the morphisms in  $Sy$  back to the identity morphisms in  $\mathcal{C}$ . We can identify  $F$  with a function  $\text{Ob } \mathcal{C} \rightarrow S$ , assigning each state in  $\mathcal{C}$  a label in  $S$  without revealing anything about the transitions between them.

- Exercise 6.63.* 1. Show that  $y$  has a unique comonoid structure.  
 2. Show that  $y$  with its comonoid structure is terminal in  $\mathbf{Cat}^\sharp$ .  
 3. Explain why  $y$  is terminal using the language of states and transitions.  $\diamond$

*Example 6.64* (Cofunctors to the walking arrow). Consider a cofunctor  $F: \mathcal{C} \rightarrow \mathcal{A}$ , where  $\mathcal{A}$  is the walking arrow category

$$\mathcal{A} := \boxed{s \xrightarrow{a} t}$$

from Example 6.34. On objects,  $F$  is a function  $\text{Ob } \mathcal{C} \rightarrow \{s, t\}$ , so each object of  $\mathcal{C}$  lies in either  $\mathcal{C}_s := F^{-1}s$  or  $\mathcal{C}_t := F^{-1}t$  (but not both). Then on morphisms, preservation of identities determines where  $F^\sharp$  sends  $\text{id}_s$  and  $\text{id}_t$ , while preservation of codomains ensures that for each  $c \in \mathcal{C}_s$ , the morphism  $F_c^\sharp a: a \rightarrow \_$  that  $F^\sharp$  sends  $a$  back to must satisfy

$$F \text{ cod } F_c^\sharp a = \text{cod } a = t$$

and thus  $\text{cod } F_c^\sharp a \in \mathcal{C}_t$ . 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  $F_c^\sharp a$ . As there are no nontrivial composites in  $\mathcal{A}$ , the cofunctor  $F$  automatically preserves composites.

In summary, a cofunctor  $F: \mathcal{C} \rightarrow \mathcal{A}$  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 cofunctor  $F: \mathcal{C} \rightarrow \mathcal{D}$  and an object  $d \in \mathcal{D}$ , we will continue to use the notation  $\mathcal{C}_d := F^{-1}d$  to denote the set of objects in  $\mathcal{C}$  that  $F$  sends to  $d$ .

*Exercise 6.65.* Let  $F: \mathcal{C} \rightarrow \mathcal{A}$  be a cofunctor from  $\mathcal{C}$  to the walking arrow category  $\mathcal{A}$ , as in Example 6.64. If  $Fc = s$  for all  $c \in \mathcal{C}$ , what can we say about  $\mathcal{C}$ ?  $\diamond$

- Exercise 6.66.* 1. Recall the star-shaped category  $y^{n+1} + ny$  from Exercise 6.37. Describe cofunctors to it.
2. Describe cofunctors to the preorder  $(\mathbb{N}, \leq)$ , viewed as a category: its objects are natural numbers, and there is a morphism  $m \rightarrow n$  if and only if  $m \leq n$ .
3. Describe cofunctors to the preorder  $(\mathbb{N}, \geq)$ : its objects are natural numbers, and there is a morphism  $n \rightarrow m$  if and only if  $n \geq m$ .  $\diamond$

*Example 6.67* (Cofunctors to the walking commutative square). Consider a cofunctor  $F: \mathcal{C} \rightarrow \mathcal{CS}$ , where  $\mathcal{CS}$  is the *walking commutative square category*

$$\mathcal{CS} := \begin{array}{ccc} w & \xrightarrow{h} & y \\ f \downarrow & & \downarrow k \\ x & \xrightarrow{g} & z \\ f \circ g & = & h \circ k \end{array}$$

On objects,  $F$  is a function  $\text{Ob } \mathcal{C} \rightarrow \{w, x, y, z\}$ , 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  $F_X^\# g: X \rightarrow \_$  to an object in  $\mathcal{C}_z$ , and out of every object  $Y \in \mathcal{C}_y$  there is a morphism  $F_Y^\# k: Y \rightarrow \_$  also to an object in  $\mathcal{C}_z$ . Finally, out of every object  $W \in \mathcal{C}_w$  there is a morphism  $F_W^\# f: W \rightarrow X_W$  to an object  $X_W \in \mathcal{C}_x$  and a morphism  $F_W^\# h: W \rightarrow Y_W$  to an object  $Y_W \in \mathcal{C}_y$ . As  $F$  preserves composites, these must all then satisfy

$$F_W^\# f \circ F_{X_W}^\# g = F_W^\# (f \circ g) = F_W^\# (h \circ k) = F_W^\# h \circ F_{Y_W}^\# k;$$

in particular,  $F_{X_W}^\# g$  and  $F_{Y_W}^\# k$  must share a common codomain  $Z_W \in \mathcal{C}_z$ , yielding the following commutative square in  $\mathcal{C}$ :

$$\begin{array}{ccc} W & \xrightarrow{F_W^\# h} & Y_W \\ F_W^\# f \downarrow & & \downarrow F_{Y_W}^\# k \\ X_W & \xrightarrow{F_{X_W}^\# g} & Z_W. \end{array}$$

*Exercise 6.68.* Let  $\mathcal{A}$  denote the walking arrow category, as in Example 6.64, and let  $\mathcal{CS}$  denote the walking commutative square category, as in Example 6.67.

1. List the cofunctors  $\mathcal{CS} \rightarrow \mathcal{A}$ .
2. List the cofunctors  $\mathcal{A} \rightarrow \mathcal{CS}$ .  $\diamond$

*Exercise 6.69.* 1. What does a cofunctor from  $y$  to a poset represent?  
 2. Consider the chain poset  $[n] \cong \sum_{i=0}^n y^{i+1}$ . How many cofunctors are there from  $[m] \rightarrow [n]$  for all  $m \leq n$ ?  $\diamond$

### Cofunctors to monoids

When a monoid  $(M, e, *)$ , viewed as a 1-object category  $y^M$ , is the codomain of a cofunctor  $\mathcal{C} \rightarrow y^M$ , it plays the role of a joystick: an “input device” that “reports its . . . direction to the device it is controlling.”<sup>9</sup> 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 6.70 (Arrow fields).* Consider the monoid  $(\mathbb{N}, 0, +)$  viewed as a category  $y^{\mathbb{N}}$ . Cofunctors  $\mathcal{C} \rightarrow y^{\mathbb{N}}$  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.<sup>10</sup> We’ll have more to say about these in Theorem 7.59, but our goal here is simply to unpack the definition.

To specify a cofunctor  $A: \mathcal{C} \rightarrow y^{\mathbb{N}}$ , 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 cofunctors to monoids,  $A$  is characterized by its behavior on morphisms: for each object  $c \in \mathcal{C}$ , the cofunctor assigns each  $n \in \mathbb{N}$  a morphism  $A_c^\# n$  of  $\mathcal{C}$  emanating from  $c$ . That’s a lot of data, but we still have two cofunctor laws to pare it down:

$$A_c^\# 0 = \text{id}_c \quad \text{and} \quad A_c^\#(m + n) = A_c^\# m \circ_{\text{cod } A_c^\# m} A_c^\# n.$$

Then for each  $c \in \mathcal{C}$ , since every  $n \in \mathbb{N}$  is a sum of 1’s, the morphism  $A_c^\# n$  can be decomposed into  $n$  copies of  $A_{c_j}^\# 1$  for objects  $c_0 := c, c_1, \dots, c_n \in \mathcal{C}$ , as follows:

$$c = c_0 \xrightarrow{A_{c_0}^\# 1} c_1 \xrightarrow{A_{c_1}^\# 1} \dots \xrightarrow{A_{c_{n-1}}^\# 1} c_n. \quad (6.71)$$

Here each  $c_{j+1} := \text{cod } A_{c_j}^\# 1$ .

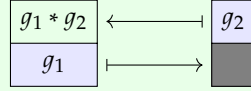
Thus an arrow field of  $\mathcal{C}$  is given by independently choosing a morphism emanating from each  $c \in \mathcal{C}$  to be  $A_c^\# 1$ : an arrow (morphism) out of each object, like how a vector field has a vector out of each point. Indeed, any such choice uniquely determines the cofunctor  $A: \mathcal{C} \rightarrow y^{\mathbb{N}}$ : 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

<sup>9</sup>Description from Wikipedia.





is a cofunctor  $Gy^G \rightarrow y^G$  given by



To see this is a cofunctor, we check that identities, codomains, and compositions are preserved. For any  $g_1$ , the identity  $e$  is passed back to  $g_1 * e = g_1$ , and this is the identity on  $g_1$  in  $Gy^G$ . Codomains are preserved because there is only one object in  $y^G$ . Composites are preserved because for any  $g_2, g_3$ , we have  $g_1 * (g_2 * g_3) = (g_1 * g_2) * g_3$ .

*Exercise 6.77.* Does the idea of Example 6.76 work when  $G$  is merely a monoid, or does something go subtly wrong somehow?  $\diamond$

**Proposition 6.78.** There is a fully faithful functor  $\mathbf{Mon}^{\text{op}} \rightarrow \mathbf{Cat}^\sharp$ , 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 cofunctor between such categories carries no data in its on-objects part, and codomains are automatically preserved. Cofunctors  $y^M \rightarrow y^N$  simply carry elements of  $N$  to elements of  $M$ , preserving identity and composition, exactly the description of monoid homomorphisms.  $\square$

**Proposition 6.79.** There is an adjunction

$$\mathbf{Cat}^\sharp(\mathcal{C}, Ay) \cong \mathbf{Set}(\text{Ob } \mathcal{C}, A)$$

for  $\mathcal{C} \in \mathbf{Cat}^\sharp$  and  $A \in \mathbf{Set}$ .

*Proof.* In the solution to Exercise 6.36, 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 cofunctor 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.  $\square$

*Exercise 6.80 (Continuous arrow fields).* Suppose we say that a *continuous arrow field* on  $\mathcal{C}$  is a cofunctor  $\mathcal{C} \rightarrow y^{\mathbb{R}}$ , 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 cofunctor is and isn't yet ready to learn.  $\diamond$

*Example 6.81* (Systems of ODEs). A system of ordinary differential equations (ODEs) in  $n$  variables, e.g.

$$\begin{aligned}\dot{x}_1 &= f_1(x_1, \dots, x_n) \\ \dot{x}_2 &= f_2(x_1, \dots, x_n) \\ &\vdots \\ \dot{x}_n &= f_n(x_1, \dots, x_n),\end{aligned}$$

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  $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ , 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

$$x^{+0} = x \quad \text{and} \quad x^{+t_1+t_2} = (x^{+t_1})^{+t_2}. \quad (6.82)$$

Let's call such things *differentiable dynamical systems* with time domain  $(T, 0, +)$ ; above, we used  $T := \mathbb{R}$ , but any monoid will do.

Dynamical systems in the above sense are cofunctors  $F: \mathbb{R}^n y^{\mathbb{R}^n} \rightarrow y^T$ . In order to say this, we first need to say how both  $\mathcal{C} := \mathbb{R}^n y^{\mathbb{R}^n}$  and  $y^T$  are being considered as categories. The category  $\mathcal{C}$  has objects  $\mathbb{R}^n$ , and for each object  $x \in \mathbb{R}^n$  and outgoing arrow  $v \in \mathbb{R}^n$ , the codomain of  $v$  is  $x + v$ ; in other words,  $v$  is a vector emanating from  $x$ . The identity is  $v = 0$ , and composition is given by addition. The category  $y^T$  is the monoid  $T$  considered as a category with one object,  $\bullet$ .

The cofunctor assigns to every object  $x \in \mathbb{R}^n$  the unique object  $F(x) = \bullet$ , and to each element  $t \in T$  the morphism  $F^\sharp(x, t) = x^{+t} - x \in \mathbb{R}^n$ , which can be interpreted as a vector emanating from  $x$ . Its codomain is  $\text{cod } F^\sharp(x, t) = x^{+t}$ , and we will see that (6.82) ensures the cofunctoriality properties.

The codomain law ii is vacuously true, since  $y^T$  only has one object. Law i follows because  $F^\sharp(x, 0) = x^{+0} - x = 0$ , and law iii follows as

$$F^\sharp(x^{+t_1}, t_2) + F^\sharp(x, t_1) = (x^{+t_1})^{+t_2} - x^{+t_1} + x^{+t_1} - x = x^{+t_1+t_2} - x = F^\sharp(x, t_1 + t_2).$$

### Cofunctors from state categories

By now we should be very familiar with lenses from state categories, which are our original dynamical systems. A cofunctor from a state category, then, is just a dynamical system that satisfies the cofunctor laws. It turns out that cofunctors from state categories are particularly noteworthy: just as a polynomial comonoid  $\mathcal{C}$  can be identified with a category, a cofunctor out of  $\mathcal{C}$  can be identified with a number of equivalent categorical constructions on  $\mathcal{C}$ , perhaps the most familiar being a functor  $\mathcal{C} \rightarrow \mathbf{Set}$ . But these equivalences deserve their own subsection to examine in full; we'll defer them to

Section 6.3.3. For now, let's look at some examples of cofunctors out of state categories.

*Example 6.83* (Cofunctors from state categories to  $\mathcal{C}$  are  $\mathcal{C}$ -coalgebras). Recall from Example 5.73 that for a set  $S$ , lenses  $Sy^S \rightarrow p$  correspond to functions  $S \rightarrow p(S)$  known as coalgebras for the functor  $p$ . As a cofunctor  $Sy^S \rightarrow \mathcal{C}$  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  $S \rightarrow \mathfrak{c}(S)$  for the functor  $\mathfrak{c}$ .

Taking  $A = B = S \in \mathbf{Set}$  in (5.71), we find that there is a natural isomorphism between dynamical systems  $Sy^S \rightarrow p$  and functions  $S \rightarrow p(S)$ , also known as a *coalgebra for the functor  $p$*  or a  *$p$ -coalgebra*.<sup>11</sup>

*Example 6.84* (Cofunctors between state categories are very well-behaved lenses). Our familiar state category on  $S$  from Example 6.38 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)  $s \circ s' = s'$  for composable  $s, s' \in S$ .

Then a cofunctor  $Sy^S \rightarrow Ty^T$  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 2.70, where we called the on-objects (on-positions) function of such a cofunctor  $\text{get}: S \rightarrow T$ , and the on-morphisms (on-directions) function  $\text{put}: S \times T \rightarrow S$ .<sup>12</sup> Then the cofunctor laws are as follows:

1. Preservation of identities (6.56) 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  $\text{get}$  before we apply  $\text{put}$ ).

2. Preservation of codomains (6.57) 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

$$\text{put}(\text{put}(s, t), t') = \text{put}(s, t')$$

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  $\text{get}$  is a product projection! For example, if the cardinalities  $|S|$  and  $|T|$  of  $S$  and  $T$  are finite and  $|S|$  is

<sup>11</sup>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.

not divisible by  $|T|$ , then there are no cofunctors  $Sy^S \rightarrow Ty^T$ . A stringent condition, no? We'll explore it in Exercise 6.86 below.

Let's explore why cofunctors between state categories are just product projections. A product projection  $A \times B \rightarrow A$  always has a second factor  $B$ ; if every cofunctor between state categories is a product projection, what is the second factor? It turns out to be

$$U := \{u: T \rightarrow S \mid \forall t, t' \in T, \text{get}(u(t)) = t \text{ and } \text{put}(u(t), t') = u(t')\}.$$

In other words, we will show that if  $(\text{get}, \text{put})$  defines a cofunctor  $Sy^S \rightarrow Ty^T$ , then there is a bijection  $S \cong T \times U$  making  $\text{get}: S \rightarrow T$  a product projection. We then prove the converse in Exercise 6.85.

Assume  $(\text{get}, \text{put}): Sy^S \rightarrow Ty^T$  is a cofunctor, so that it satisfies the enumerated laws. First, we define a function  $\alpha: S \rightarrow T \times U$  as follows. Given  $s \in S$ , the function  $u_s: T \rightarrow S$  defined by

$$u_s(t) = \text{put}(s, t)$$

lies in  $U$ : we check that it satisfies

$$\text{get}(u_s(t)) = \text{get}(\text{put}(s, t)) = t$$

by the put-get law for  $t \in T$  and

$$\text{put}(u_s(t), t') = \text{put}(\text{put}(s, t), t') = \text{put}(s, t') = u_s(t')$$

by the put-put law for  $t, t' \in T$ . We can therefore define a function  $\alpha: S \rightarrow T \times U$  by

$$\alpha(s) = (\text{get}(s), u_s).$$

In the other direction, we have a function  $\beta: T \times U \rightarrow S$  given by

$$\beta(t, u) = u(t).$$

The two functions  $\alpha$  and  $\beta$  are mutually inverse:  $\alpha \circ \beta: S \rightarrow S$  is the identity because

$$\beta(\alpha(s)) = u_s(\text{get}(s)) = \text{put}(s, \text{get}(s)) = s$$

by the get-put law, while  $\beta \circ \alpha: T \times U \rightarrow T \times U$  is the identity because

$$\alpha(\beta(t, u)) = (\text{get}(u(t)), u_{u(t)}) = (t, t' \mapsto \text{put}(u(t), t')) = (t, u),$$

as  $\text{get}(u(t)) = t$  and  $\text{put}(u(t), t') = u(t')$  by construction for  $u \in U$ . Thus  $S \cong T \times U$ , and the product projection  $S \xrightarrow{\alpha} T \times U \rightarrow T$  sends  $s \mapsto \text{get}(s)$ , as desired.

We have therefore shown that for every cofunctor  $Sy^S \rightarrow Ty^T$ , there exists a set  $U$  for which  $S \cong T \times U$  and the on-positions function  $\text{get}$  is the product projection  $S \cong T \times U \rightarrow T$ . Notice that the on-directions function  $\text{put}$  can be uniquely recovered

from the bijection  $S \cong T \times U$  we constructed: it is determined by the functions  $u_s : T \rightarrow S$  for  $s \in S$ , which in turn is determined by the second projection  $T \times U \rightarrow U$ .

More precisely, composing  $\alpha : S \rightarrow T \times U$  with the projection to  $U$  yields a map  $S \rightarrow U$  sending  $s \mapsto u_s$ ; then  $\text{put}(s, t)$  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  $T \rightarrow S$ . This is where  $\beta : T \times U \rightarrow S$  comes in: we know  $\beta(t, u_s) = u_s(t)$ . So  $\text{put}(s, t)$  must be  $\beta(t, u_s)$ .

*Exercise 6.85.* Let  $S, T, U$  be sets for which we have a bijection  $S \cong T \times U$ . Show that there exists a unique cofunctor  $Sy^S \rightarrow Ty^T$  whose on-positions function  $S \cong T \times U \rightarrow T$  is given by the product projection.  $\diamond$

*Exercise 6.86.* 1. Suppose  $|S| = 3$ . How many cofunctors are there  $Sy^S \rightarrow Sy^S$ ?  
2. Suppose  $|S| = 4$  and  $|T| = 2$ . How many cofunctors are there  $Sy^S \rightarrow Ty^T$ ?  $\diamond$

*Example 6.87.* We have a state category  $\mathfrak{c}(1)y^{\mathfrak{c}(1)}$  on the set of objects of  $\mathcal{C}$ . Define a lens  $\mathfrak{c}(1)y^{\mathfrak{c}(1)} \rightarrow \mathfrak{c}$  by

$$\begin{array}{ccc} \text{Ob } \mathcal{C} & \boxed{\text{cod } f} & \xleftarrow{\text{cod}} \boxed{f} \text{ } \mathcal{C}[-] \\ \text{Ob } \mathcal{C} & \boxed{i} & \xlongequal{\quad} \boxed{i} \text{ } \text{Ob } \mathcal{C} \\ \mathfrak{c}(1)y^{\mathfrak{c}(1)} & & \mathfrak{c} \end{array}$$

sending each object  $i \in \mathcal{C}$  to itself on positions and, at  $i$ , sending each morphism  $f : i \rightarrow \_$  to its codomain  $\text{cod } f$  on directions.

This lens is a cofunctor  $\mathfrak{c}(1)y^{\mathfrak{c}(1)} \rightarrow \mathcal{C}$  because it sends identities back to identities, codomains forward to codomains, and preserves composition (trivially, since each morphism in  $\mathfrak{c}(1)y^{\mathfrak{c}(1)}$  is determined by its domain and codomain).

*Exercise 6.88.* Fix an object  $i \in \mathfrak{c}(1) = \text{Ob } \mathcal{C}$ . 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  $\mathfrak{c}[i]y^{\mathfrak{c}[i]} \rightarrow \mathfrak{c}$  by

$$\begin{array}{ccc} \mathcal{C}[i] & \boxed{f \circ g} & \xleftarrow{\circ} \boxed{g} \text{ } \mathcal{C}[-] \\ \mathcal{C}[i] & \boxed{f} & \xrightarrow{\text{cod}} \boxed{\text{cod } f} \text{ } \text{Ob } \mathcal{C} \\ \mathfrak{c}[i]y^{\mathfrak{c}[i]} & & \mathfrak{c} \end{array}$$

sending each morphism  $f : i \rightarrow \_$  to its codomain on positions and, at  $f$ , sending each morphism  $g : \text{cod } f \rightarrow \_$  to the composite  $f \circ g : i \rightarrow \_$  on directions. Is this lens a

<sup>12</sup>More precisely, we are treating the on-morphism functions  $T \rightarrow S$  for each  $s \in S$  of a cofunctor  $Sy^S \rightarrow Ty^T$  as a single function  $S \times T \rightarrow S$ .

cofunctor  $c[i]y^{c[i]} \rightarrow C?$

◇

We'll revisit cofunctors from state categories in Section 6.3.3.

### Other cofunctors

*Example 6.89* (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  $\mathcal{T} \rightarrow \mathcal{C}$  naturally correspond to objects in  $\mathcal{C}$ .

Unfortunately, the same cannot be said for cofunctors: we'll see in Exercise 6.90 that there does not exist a fixed category  $\mathcal{U}$  for which cofunctors  $\mathcal{U} \rightarrow \mathcal{C}$  are in bijection with objects in  $\mathcal{C}$  for every category  $\mathcal{C}$ .

Cofunctors  $\mathcal{T} \rightarrow \mathcal{C}$  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  $c \rightarrow c'$  is  $c' = c$  itself.

*Exercise 6.90.* We saw in Exercise 6.27 that  $2y$  has a unique comonoid structure.

1. Show that for any category  $\mathcal{U}$ , cofunctors  $\mathcal{U} \rightarrow 2y$  are in bijection with the set  $2^{\text{Ob } \mathcal{U}}$ .
2. Use the case of  $\mathcal{C} := 2y$  to show that if cofunctors  $\mathcal{U} \rightarrow \mathcal{C}$  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 cofunctors  $\mathcal{U} \rightarrow \mathcal{D}$  are in bijection with objects in  $\mathcal{D}$ , then  $\mathcal{U}$  must have more than one object. Conclude that objects are not representable in  $\mathbf{Cat}^\sharp$  the way they are in  $\mathbf{Cat}$ .
4. Is there a fixed category  $\mathcal{V}$  for which cofunctors  $\mathcal{E} \rightarrow \mathcal{V}$  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 6.91.* Consider the category  $\mathbb{R}y^\mathbb{R}$ , where the codomain of  $r$  emanating from  $x$  is  $x + r$ , identities are 0, and composition is given by addition. What are cofunctors into  $\mathbb{R}y^\mathbb{R}$ ?

Let  $\mathcal{C}$  be a category and  $|\cdot|: \mathcal{C} \rightarrow \mathbb{R}y^\mathbb{R}$  a cofunctor. It assigns to every object  $c$  both a real number  $|c| \in \mathbb{R}$  and a choice of emanating morphism  $|c|^\sharp(r): c \rightarrow c_r$  such that  $|c| + r = |c_r|$ . This assignment satisfies some laws. Namely we have  $c_0 = c$  and, given reals  $r, s \in \mathbb{R}$ , we have  $(c_r)_s = c_{r+s}$ .

*Exercise 6.92.* How many cofunctors

$$\boxed{s \xrightarrow{a} t} \rightarrow \boxed{u \xrightleftharpoons[c]{b} v}$$

are there from the walking arrow category  $\mathcal{A}$ , drawn above on the left, to the walking parallel-arrows category  $\mathcal{P}\mathcal{A}$ , drawn above on the right?  $\diamond$

- Exercise 6.93.*
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 cofunctor  $\mathcal{C} \rightarrow \mathcal{D}$  a cofunctor  $\mathfrak{c}y \rightarrow \mathfrak{d}y$  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?  $\diamond$

*Exercise 6.94.* Suppose  $\mathfrak{c}, \mathfrak{d}, \mathfrak{e}$  are polynomials, each with a comonoid structure, and that  $f: \mathfrak{c} \rightarrow \mathfrak{d}$  and  $g: \mathfrak{d} \rightarrow \mathfrak{e}$  are lenses.

1. If  $f$  and  $f \circ g$  are each cofunctors, is  $g$  automatically a cofunctor? If so, sketch a proof; if not, sketch a counterexample.
2. If  $g$  and  $f \circ g$  are each cofunctors, is  $f$  automatically a cofunctor? If so, sketch a proof; if not, sketch a counterexample.  $\diamond$

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 cofunctors along the way. But first, we'll conclude this chapter with several alternative characterizations of cofunctors out of state categories.

### 6.3.3 Equivalent characterizations of cofunctors from state categories

Fix a category  $\mathcal{C}$  throughout with polynomial carrier  $\mathfrak{c}$ . How can we view the data of a cofunctor from a state category  $Sy^S \rightarrow \mathcal{C}$ ? 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 5.73 that for a set  $S$ , lenses  $Sy^S \rightarrow p$  correspond to functions  $S \rightarrow p(S)$  known as coalgebras for the functor  $p$ . As a cofunctor  $Sy^S \rightarrow \mathcal{C}$  is just a special kind of lens from  $Sy^S$  to  $\mathfrak{c}$ , it should correspond to a special kind of coalgebra  $S \rightarrow \mathfrak{c}(S)$  for the functor  $\mathfrak{c}$ . Indeed, whenever  $\mathfrak{c}$  carries a comonoid  $\mathcal{C}$  with respect to the composition product (i.e. a comonad), there is a special notion of a  $\mathcal{C}$ -coalgebra (i.e. a coalgebra for the comonad  $\mathcal{C}$ ), as follows.

**Definition 6.95** (Coalgebra for a polynomial comonoid). Let  $\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$  be a polynomial comonoid. A  $\mathcal{C}$ -coalgebra  $(S, \alpha)$  is

- a set  $S$ , called the *carrier*, equipped with
- a function  $\alpha: S \rightarrow \mathfrak{c} \triangleleft S$ ,

such that the following diagrams, collectively known as the *coalgebra laws*, commute:

$$\begin{array}{ccc}
 S & \xrightarrow{\alpha} & \mathfrak{c} \triangleleft S \\
 \searrow & & \downarrow \epsilon \triangleleft S \\
 & & S
 \end{array}
 \qquad
 \begin{array}{ccc}
 S & \xrightarrow{\alpha} & \mathfrak{c} \triangleleft S \\
 \downarrow \alpha & & \downarrow \delta \triangleleft S \\
 \mathfrak{c} \triangleleft S & \xrightarrow{\mathfrak{c} \triangleleft \alpha} & \mathfrak{c} \triangleleft \mathfrak{c} \triangleleft S.
 \end{array}
 \tag{6.96}$$

A *morphism* of  $\mathcal{C}$ -coalgebras  $(S, \alpha) \rightarrow (T, \beta)$  is a function  $h: S \rightarrow T$  such that the following diagram commutes:

$$\begin{array}{ccc}
 S & \xrightarrow{\alpha} & \mathfrak{c} \triangleleft S \\
 h \downarrow & & \downarrow \mathfrak{c} \triangleleft h \\
 T & \xrightarrow{\beta} & \mathfrak{c} \triangleleft T
 \end{array}$$

**Proposition 6.97.** Cofunctors  $Sy^S \nrightarrow \mathcal{C}$  can be identified (up to isomorphism) with  $\mathcal{C}$ -coalgebras carried by  $S$ .

*Proof.* Let  $\mathfrak{c}$  be the carrier of  $\mathcal{C}$ . In Example 5.73, we showed that (5.71) gives a natural correspondence between lenses  $\Phi: Sy^S \rightarrow \mathfrak{c}$  and functions  $\varphi: S \rightarrow \mathfrak{c} \triangleleft S$ . We can unravel this correspondence via the proof of Proposition 5.63 as follows. A lens  $\Phi: Sy^S \rightarrow \mathfrak{c}$  can be drawn like so (we will adopt our former convention of identifying each morphism  $s \rightarrow t$  from  $Sy^S$  with its codomain  $t$ ):

$$\begin{array}{ccc}
 Sy^S & \xleftarrow{\Phi^\#} & \boxed{f} \\
 \downarrow \Phi_1 & & \downarrow i \\
 \boxed{s} & & \boxed{i}
 \end{array}
 \quad \mathfrak{c}$$

Meanwhile, the corresponding function  $\varphi: S \rightarrow \mathfrak{c} \triangleleft S$ , 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):

$$\begin{array}{ccc}
 S & & S \\
 \boxed{s} & \xrightarrow{\Phi^\#} & \boxed{t} \\
 \downarrow \Phi_1 & & \downarrow f \\
 \boxed{s} & & \boxed{f} \\
 & & \downarrow i \\
 & & \boxed{i}
 \end{array}
 \quad \mathfrak{c}$$

Then it suffices to show that  $\Phi$  satisfies the cofunctor laws if and only if  $\varphi$  satisfies the coalgebra laws. We can verify this using polyboxes. From (6.53), the eraser preservation law for  $\Phi$  would state the following (remember that the arrow in the eraser for the state



category  $Sy^S$  is just an equality):

$$Sy^S \begin{array}{|c|} \hline s \\ \hline s \\ \hline \end{array} \text{idy} = Sy^S \begin{array}{|c|} \hline \phantom{s} \\ \hline s \\ \hline \end{array} \begin{array}{c} \xleftarrow{\Phi^\#} \\ \xrightarrow{\Phi_1} \end{array} \begin{array}{|c|} \hline \phantom{s} \\ \hline \phantom{s} \\ \hline \end{array} \begin{array}{c} \text{idy} \\ \text{c} \end{array}$$

Meanwhile, the commutative triangle on the left of (6.96) can be written as follows:

$$S \begin{array}{|c|} \hline \phantom{s} \\ \hline s \\ \hline \end{array} = S \begin{array}{|c|} \hline \phantom{s} \\ \hline s \\ \hline \end{array} \begin{array}{c} \xrightarrow{\Phi^\#} \\ \xrightarrow{\Phi_1} \end{array} \begin{array}{|c|} \hline \phantom{s} \\ \hline \phantom{s} \\ \hline \end{array} \begin{array}{c} S \\ \text{idy} \\ \text{c} \end{array}$$

But these polybox equations are entirely equivalent.

Then from (6.54), the duplicator preservation law for  $\Phi$  would state the following (remember that the three arrows in the duplicator for the state category  $Sy^S$  are all equalities)

$$Sy^S \begin{array}{|c|} \hline \phantom{s} \\ \hline s \\ \hline \end{array} \begin{array}{c} \text{run} \\ \text{tgt} \end{array} \begin{array}{c} Sy^S \\ \xleftarrow{\Phi^\#} \\ \xrightarrow{\Phi_1} \end{array} \begin{array}{|c|} \hline \phantom{s} \\ \hline h \\ \hline \end{array} \begin{array}{c} c \\ \phantom{c} \end{array} = Sy^S \begin{array}{|c|} \hline \phantom{s} \\ \hline s \\ \hline \end{array} \begin{array}{c} \xleftarrow{\Phi^\#} \\ \xrightarrow{\Phi_1} \end{array} \begin{array}{|c|} \hline \phantom{s} \\ \hline \phantom{s} \\ \hline \end{array} \begin{array}{c} c \\ \text{cod} \\ \text{c} \end{array}$$

Meanwhile, the commutative triangle on the right of (6.96) can be written as follows:

$$Sy^S \begin{array}{|c|} \hline \phantom{s} \\ \hline s \\ \hline \end{array} \begin{array}{c} \xrightarrow{\Phi^\#} \\ \xrightarrow{\Phi_1} \end{array} \begin{array}{|c|} \hline \phantom{s} \\ \hline h \\ \hline \end{array} \begin{array}{c} Sy^S \\ \xleftarrow{\Phi^\#} \\ \xrightarrow{\Phi_1} \end{array} \begin{array}{|c|} \hline \phantom{s} \\ \hline \phantom{s} \\ \hline \end{array} \begin{array}{c} c \\ \text{cod} \\ \text{c} \end{array} = Sy^S \begin{array}{|c|} \hline \phantom{s} \\ \hline s \\ \hline \end{array} \begin{array}{c} \xrightarrow{\Phi^\#} \\ \xrightarrow{\Phi_1} \end{array} \begin{array}{|c|} \hline \phantom{s} \\ \hline \phantom{s} \\ \hline \end{array} \begin{array}{c} c \\ \text{cod} \\ \text{c} \end{array}$$

But these polybox equations are equivalent as well. Hence the cofunctor laws for  $\Phi$  are equivalent to the coalgebra laws for  $\varphi$ .  $\square$

So a cofunctor 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 6.98** (Discrete opfibration). Let  $\mathcal{C}$  be a category. A pair  $(\mathcal{S}, \pi)$ , where  $\mathcal{S}$  is a category and  $\pi: \mathcal{S} \rightarrow \mathcal{C}$  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  $f: \pi(s) \rightarrow c'$  there exists a unique object  $s' \in \mathcal{S}$  and morphism  $\bar{f}: s \rightarrow s'$  such that  $\pi(\bar{f}) = f$ . Note that in this case  $\pi(s') = c$ .

$$\begin{array}{ccc} s & \xrightarrow{\bar{f}} & s' \\ \pi \downarrow & & \downarrow \pi \\ \pi(s) & \xrightarrow{f} & - \end{array}$$

A *morphism* of discrete opfibrations  $(\mathcal{S}, \pi) \rightarrow (\mathcal{S}', \pi')$  over  $\mathcal{C}$  is a functor  $F: \mathcal{S} \rightarrow \mathcal{S}'$  making the following triangle commute: We refer to  $\bar{f}$  as the *lift* of  $f$  to  $s$ .

A *morphism*  $(\mathcal{S}, \pi) \rightarrow (\mathcal{S}', \pi')$  between discrete opfibrations over  $\mathcal{C}$  is a functor  $F: \mathcal{S} \rightarrow \mathcal{S}'$  making the following triangle commute:

$$\begin{array}{ccc} \mathcal{S} & \xrightarrow{F} & \mathcal{S}' \\ \pi \searrow & & \swarrow \pi' \\ & \mathcal{C} & \end{array} \quad (6.99)$$

We denote the category of discrete opfibrations over  $\mathcal{C}$  by  $\mathbf{dopf}(\mathcal{C})$ .

*Exercise 6.100.* Show that if  $F: \mathcal{S} \rightarrow \mathcal{S}'$  is a functor making the triangle (6.99) commute, where both  $\pi$  and  $\pi'$  are discrete opfibrations, then  $F$  is also a discrete opfibration.  $\diamond$

*Exercise 6.101.* Suppose  $\pi: \mathcal{S} \rightarrow \mathcal{C}$  is a discrete opfibration and  $i \in \mathcal{S}$  is an object. With notation as in Definition 6.98, show the following:

1. Show that the lift  $\overline{\text{id}_{\pi(i)}} = \text{id}_i$  of the identity on  $\pi(i)$  is the identity on  $i$ .
2. Show that for  $f: \pi(i) \rightarrow c$  and  $g: c \rightarrow c'$ , we have  $\bar{f} \circ \bar{g} = \overline{f \circ g}$ .
3. Show how  $\pi$  could instead be interpreted as a cofunctor.  $\diamond$

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 6.102.** The category of  $\mathcal{C}$ -coalgebras is isomorphic to the category  $\mathbf{dopf}(\mathcal{C})$  of discrete opfibrations over  $\mathcal{C}$ .

*Proof.* \*\*

□

### 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  $\mathcal{C} \rightarrow \mathbf{Set}$ . Such a functor is known as a *copresheaf* on  $\mathcal{C}$  for short. Here we review what is needed to understand this equivalence. We begin by giving a standard construction on any copresheaf.

**Definition 6.103** (Category of elements). Given a copresheaf on  $\mathcal{C}$ , i.e. a functor  $I: \mathcal{C} \rightarrow \mathbf{Set}$ , its *category of elements*  $\int^{\mathcal{C}} I$  is defined to have objects

$$\mathrm{Ob} \int^{\mathcal{C}} I := \{(c, x) \mid c \in \mathcal{C}, x \in Ic\}$$

and a morphism  $f: (c, x) \rightarrow (c', x')$  for every morphism  $f: c \rightarrow c'$  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  $f: c \rightarrow c'$  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 6.104.* Let  $I: \mathcal{C} \rightarrow \mathbf{Set}$  be a functor, and let  $\int^{\mathcal{C}} I$  be as in Definition 6.103.

1. Show that there is a functor  $\pi: \int^{\mathcal{C}} I \rightarrow \mathcal{C}$  sending objects  $(c, x) \mapsto c$  and morphisms  $f: (c, x) \rightarrow (c', x')$  to  $f: c \rightarrow c'$ .
2. Show that  $\pi$  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 6.105.* Suppose that  $I, J: \mathcal{C} \rightarrow \mathbf{Set}$  are functors and  $\alpha: I \rightarrow J$  is a natural transformation.

1. Show that  $\alpha$  induces a functor  $(\int^{\mathcal{C}} I) \rightarrow (\int^{\mathcal{C}} J)$ .
2. Show that it is a morphism of discrete opfibrations in the sense of Definition 6.98.
3. Have you now verified that there is a functor

$$\int^{\mathcal{C}}: \mathbf{Set}^{\mathcal{C}} \rightarrow \mathbf{dopf}(\mathcal{C})$$

or is there something left to do?

◇

*Exercise 6.106.* Let  $G$  be a graph, and let  $\mathcal{G}$  be the free category on it. Show that for any functor  $S: \mathcal{G} \rightarrow \mathbf{Set}$ , the category  $\int^{\mathcal{G}} S$  of elements is again free on a graph. ◇

**Proposition 6.107.** 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 6.105 we have a functor  $\int^{\mathcal{C}}: \mathbf{Set}^{\mathcal{C}} \rightarrow \mathbf{dopf}(\mathcal{C})$ . There is a functor going back: given a discrete opfibration  $\pi: \mathcal{S} \rightarrow \mathcal{C}$ , we define a functor  $\partial\pi: \mathcal{C} \rightarrow \mathbf{Set}$  on objects by sending each  $c \in \mathcal{C}$  to the set of objects in  $\mathcal{S}$  that  $\pi$  maps to  $c$ ; that is,

$$(\partial\pi)(c) := \{s \in \mathcal{S} \mid \pi(s) = c\}.$$

Then on morphisms, for each  $f: c \rightarrow _$  in  $\mathcal{C}$  and  $s \in (\partial\pi)(c)$  we have  $\pi(s) = c$ , so by Definition 6.98 there exists a unique morphism  $\bar{f}: s \rightarrow _$  for which  $\pi(\bar{f}) = f$ . As  $\pi(\text{cod } \bar{f}) = \text{cod } f$ , we have  $\text{cod } \bar{f} \in (\partial\pi)(\text{cod } f)$ , so we can define

$$(\partial\pi)(f)(s) := \text{cod } \bar{f}.$$

On objects, the roundtrip  $\mathbf{Set}^{\mathcal{C}} \rightarrow \mathbf{Set}^{\mathcal{C}}$  sends  $I: \mathcal{C} \rightarrow \mathbf{Set}$  to the functor

$$\begin{aligned} c &\mapsto \{s \in \int^{\mathcal{C}} I \mid \pi(s) = c\} \\ &= \{(c, x) \mid x \in I(c)\} &= I(c). \end{aligned}$$

The roundtrip  $\mathbf{dopf}(\mathcal{C}) \rightarrow \mathbf{dopf}(\mathcal{C})$  sends  $\pi: \mathcal{S} \rightarrow \mathcal{C}$  to the discrete opfibration whose object set is  $\{(c, s) \in \text{Ob}(\mathcal{C}) \times \text{Ob}(\mathcal{S}) \mid \pi(s) = c\}$  and this set is clearly in bijection with  $\text{Ob}(\mathcal{S})$ . Proceeding similarly, one defines an isomorphism of categories  $\mathcal{S} \cong \int^{\mathcal{C}} \partial\pi$ . □

**Proposition 6.108.** 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  $\pi: \mathcal{S} \rightarrow \mathcal{C}$ , take  $S := \text{Ob}(\mathcal{S})$  and define  $(\varphi_1, \varphi^\sharp): Sy^S \rightarrow \mathfrak{c}$  by  $\varphi_1 = \pi$  and with  $\varphi^\sharp$  given by the lifting:  $\varphi(g) := \hat{g}$  as in Definition 6.98. One checks using Exercise 6.101 that this defines a cofunctor.

Conversely, given a cofunctor  $(\varphi_1, \varphi^\sharp): Sy^S \rightarrow \mathfrak{c}$ , the function  $\varphi_1$  induces a map of polynomials  $Sy \rightarrow \mathfrak{c}$ , and we can factor it as a vertical followed by a cartesian  $Sy \rightarrow \mathfrak{s} \xrightarrow{\psi} \mathfrak{c}$ . We can give  $\mathfrak{s}$  the structure of a category such that  $\psi$  is a cofunctor; see Exercise 6.109. □

*Exercise 6.109.* With notation as in Proposition 6.108, complete the proof as follows.

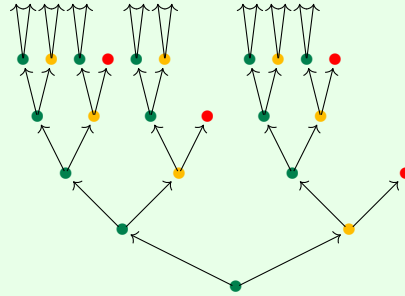
1. Check that  $(\varphi, \varphi^\sharp)$  defined in the first paragraph is indeed a cofunctor.
2. Find a comonoid structure on  $\mathfrak{s}$  such that  $\psi$  is a cofunctor, as stated in the second paragraph.
3. Show that the two directions are inverse, up to isomorphism.  $\diamond$

*Example 6.110.* In Example 7.50 we had a dynamical system with  $S := \{\bullet, \bullet, \bullet\}$  and  $p := y^2 + 1$ , and  $\varphi: Sy^S \rightarrow p$  from Exercise 3.21, depicted here again for your convenience:



Under the forgetful-cofree adjunction, the lens  $\varphi$  coincides with a cofunctor  $F: Sy^S \rightarrow \mathcal{T}_p$  from the state category on  $S$  to the category of  $p$ -trees. We can now see this as a copresheaf on the category  $\mathcal{T}_p$  itself.

The cofree category  $\mathcal{T}_i$  is actually the free category on a graph, as we saw in Proposition 7.55, and so the schema is easy. There is one table for each tree (object in  $\mathcal{T}_p$ ), 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  $Sy^S \rightarrow p$ , we extend it to a cofunctor  $\varphi: Sy^S \rightarrow \mathcal{T}_p$ . By Propositions 6.107 and 6.108, we can consider it as a discrete opfibration over  $\mathcal{T}_p$ . By Exercise 6.106 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 (6.111).

*Exercise 6.112.* Give an example of a dynamical system on  $p := y^2 + y$  for which the corresponding copresheaf has in its image a set with at least two elements.  $\diamond$

*Exercise 6.113.* Given a cofunctor  $F: Sy^S \rightarrow y$ , what does the corresponding copresheaf look like?  $\diamond$

To summarize, we have four equivalent notions:

- (1) cofunctors  $F: Sy^S \rightarrow \mathcal{C}$ ;
- (2)  $\mathcal{C}$ -coalgebras  $(S, \alpha)$ , with  $\alpha: S \rightarrow c \triangleleft S$ ;
- (3) discrete opfibrations  $\pi: \mathcal{S} \rightarrow \mathcal{C}$ , with  $\text{Ob } \mathcal{S} = S$ ;
- (4) copresheaves  $I: \mathcal{C} \rightarrow \mathbf{Set}$ , with  $\text{Ob } \int^{\mathcal{C}} I = S$ .

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 7.66 and ??).

## 6.4 Summary and further reading

In this chapter we began by showing that for every set  $S$ , the thing that makes a dynamical system like  $Sy^S \rightarrow p$  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  $\epsilon: p \rightarrow y$  and  $\delta: p \rightarrow p \triangleleft p$ , are exactly categories [AU16]. We explained how  $\epsilon$  picks out an identity for each object and how  $\delta$  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  $c \rightarrow d$  are not functors but *cofunctors*: they map forwards on objects and backwards on morphisms. Cofunctors were first defined by Marcelo Aguiar [Agu97], though his definition was opposite to ours. Cofunctors are the morphisms of a category we notate as  $\mathbf{Cat}^\#$ . We showed that cofunctors between state categories  $Sy^S \rightarrow Ty^T$  are what are known in the functional programming community as *very well-behaved lenses*. For more on lenses, see [nLa22].

## 6.5 Exercise solutions

*Solution to Exercise 6.2.*

Given a polynomial  $s \in \mathbf{Poly}$  equipped with a lens  $\epsilon: s \rightarrow y$ , we know that  $\epsilon$  picks out a direction at every position of  $s$ . So all we can say about  $s$  is that there is at least one direction at each of its positions. Equivalently, we could say that  $s$  can be written as the product of  $y$  and some other polynomial.

*Solution to Exercise 6.9.*

Following the arrows on either side of (6.8) 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  $s_0 \rightarrow s_3$  and follow the arrows backward, unpacking each term until only the contents of the blue boxes remain (namely  $s_0, s_0 \rightarrow s_1, s_1 \rightarrow s_2$ , and  $s_2 \rightarrow s_3$ ). Here's what we get when we follow this process for the left hand side of (6.8) (remember where to look for the three inputs to the run function):

$$\begin{aligned} s_0 \rightarrow s_3 &= \text{run}(s_0, s_0 \rightarrow s_2, s_2 \rightarrow s_3) \\ &= \text{run}(s_0, \text{run}(s_0, s_0 \rightarrow s_1, s_1 \rightarrow s_2), s_2 \rightarrow s_3) \\ &= \text{run}(s_0, \text{run}(s_0, s_0 \rightarrow s_1, s_1 \rightarrow s_2), s_2 \rightarrow s_3); \end{aligned}$$

and here's what we get for the right (also remember where to look for the two inputs to the target function):

$$\begin{aligned} s_0 \rightarrow s_3 &= \text{run}(s_0, s_0 \rightarrow s_1, s_1 \rightarrow s_3) \\ &= \text{run}(s_0, s_0 \rightarrow s_1, \text{run}(s_1, s_1 \rightarrow s_2, s_2 \rightarrow s_3)) \\ &= \text{run}(s_0, s_0 \rightarrow s_1, \text{tgt}(s_0, s_0 \rightarrow s_1), s_1 \rightarrow s_2, s_2 \rightarrow s_3)). \end{aligned}$$

Setting these equal yields our desired associativity equation:

$$\text{run}(s_0, \text{run}(s_0, s_0 \rightarrow s_1, s_1 \rightarrow s_2), s_2 \rightarrow s_3) = \text{run}(s_0, s_0 \rightarrow s_1, \text{tgt}(s_0, s_0 \rightarrow s_1), s_1 \rightarrow s_2, s_2 \rightarrow s_3)).$$

*Solution to Exercise 6.11.*

1. Given  $s \in \mathbf{Poly}$  and a lens  $\delta: s \rightarrow s \triangleleft s$ , we want all the ways to obtain a lens  $s \rightarrow s^{\triangleleft 4}$  using  $\delta$ ,  $\text{id}_s$  (i.e.  $s$ ),  $\triangleleft$ , and  $\circ$ . Starting with  $s$ , the only way to get to  $s^{\triangleleft 2}$  is with a single  $\delta: s \rightarrow s^{\triangleleft 2}$ . From there, we can get to  $s^{\triangleleft 4}$  directly by composing with  $\delta \triangleleft \delta$  to obtain  $\delta \circ (\delta \triangleleft \delta)$ . Alternatively, we can preserve either the first or the second  $s$  using the identity, then get to  $s^{\triangleleft 3}$  from the other  $s$  in one of two ways: either  $\delta \circ (\delta \triangleleft s)$  or  $\delta \circ (s \triangleleft \delta)$ . This gives us 4 more ways to write a lens  $s \rightarrow s^{\triangleleft 4}$ : either  $\delta \circ (s \triangleleft (\delta \circ (\delta \triangleleft s)))$  or  $\delta \circ (s \triangleleft (\delta \circ (s \triangleleft \delta)))$  if we chose to preserve the first  $s$ , and either  $\delta \circ ((\delta \circ (\delta \triangleleft s)) \triangleleft s)$  or  $\delta \circ ((\delta \circ (s \triangleleft \delta)) \triangleleft s)$  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  $\delta$ :

- (1)  $\delta \circ ((\delta \circ (\delta \triangleleft s)) \triangleleft s)$
- (2)  $\delta \circ ((\delta \circ (s \triangleleft \delta)) \triangleleft s)$
- (3)  $\delta \circ (\delta \triangleleft \delta)$
- (4)  $\delta \circ (s \triangleleft (\delta \circ (\delta \triangleleft s)))$
- (5)  $\delta \circ (s \triangleleft (\delta \circ (s \triangleleft \delta)))$

This coincides with the 5 different ways to parenthesize a 4-term expression.

2. We wish to show that if (6.10) commutes, then all the lenses on our list are equal. The commutativity of (6.10) implies that  $\delta \circ (\delta \triangleleft s) = \delta \circ (s \triangleleft \delta)$ ; so (1) and (2) from our list are equal, as are (4) and (5). Meanwhile, since  $s = s \circ s$ , we can rewrite (1) as

$$\delta \circ ((\delta \circ (\delta \triangleleft s)) \triangleleft (s \circ s)) = \delta \circ (\delta \triangleleft s) \circ (\delta \triangleleft s \triangleleft s),$$

where the associativity of  $\triangleleft$  and  $\circ$  allows us to drop some parentheses. Then the commutativity of (6.10) allows us to further rewrite this as

$$\begin{aligned} \delta \circ (s \triangleleft \delta) \circ (\delta \triangleleft s \triangleleft s) &= \delta \circ ((s \circ \delta) \triangleleft (\delta \circ (s \triangleleft s))) \\ &= \delta \circ (\delta \triangleleft \delta), \end{aligned}$$

so (1) and (3) are equal. Similarly, we can rewrite (5) as

$$\begin{aligned} \delta \circ ((s \circ s) \triangleleft (\delta \circ (s \triangleleft \delta))) &= \delta \circ (s \triangleleft \delta) \circ (s \triangleleft s \triangleleft \delta) \\ &= \delta \circ (\delta \triangleleft s) \circ (s \triangleleft s \triangleleft \delta) \end{aligned}$$

$$\begin{aligned}
&= \delta \circ ((\delta \circ (\delta \circ \delta)) \triangleleft (\delta \circ \delta)) \\
&= \delta \circ (\delta \triangleleft \delta)
\end{aligned}$$

so (3) and (5) are equal. Hence all the lenses on our list are equal.

*Solution to Exercise 6.12.*

We have  $\text{Run}_n(\varphi) = \delta^{(n)} \circ \varphi^{\triangleleft n}$  for all  $n \in \mathbb{N}$ , as well as  $\delta^{(0)} = \epsilon$  and  $\delta^{(1)} = \text{id}_s$ . Then  $\text{Run}_0(\varphi) = \epsilon \circ \varphi^{\triangleleft 0} = \epsilon \circ \text{id}_y = \epsilon$  and  $\text{Run}_1(\varphi) = \text{id}_s \circ \varphi^{\triangleleft 1} = \varphi^{\triangleleft 1} = \varphi$ .

*Solution to Exercise 6.21.*

We complete the proof of Proposition 6.20, where we are given a comonoid  $(c, \epsilon, \delta)$  along with  $\delta^{(0)} := \epsilon$  and  $\delta^{(n+1)} := \delta \circ (\delta^{(n)} \triangleleft c)$  for all  $n \in \mathbb{N}$ .

1. We will show that  $\delta^{(n)}$  is a map  $c \rightarrow c^{\triangleleft n}$  for every  $n \in \mathbb{N}$  by induction on  $n$ . We know  $\delta^{(0)} = \epsilon$  is a map  $c \rightarrow y = c^{\triangleleft 0}$ , and for each  $n \in \mathbb{N}$ , if  $\delta^{(n)}$  is a map  $c \rightarrow c^{\triangleleft n}$ , then the composite  $\delta^{(n+1)} = \delta \circ (\delta^{(n)} \triangleleft c)$  is a map

$$c \xrightarrow{\delta} c \triangleleft c \xrightarrow{\delta^{(n)} \triangleleft c} c^{\triangleleft n} \triangleleft c \cong c^{\triangleleft (n+1)}$$

Hence the result follows by induction.

2. We have  $\delta^{(1)} = \delta \circ (\delta^{(0)} \triangleleft c) = \delta \circ (\epsilon \triangleleft c) = \text{id}_c$  by the left erasure law from (6.15).
3. By the previous part, we have  $\delta^{(2)} = \delta \circ (\delta^{(1)} \triangleleft c) = \delta \circ (c \triangleleft c) = \delta$ .

*Solution to Exercise 6.26.*

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 (6.24). The erasure (counit) laws are verified by looking at the picture of  $\delta$  in (6.25) and noting that it sends the double (do-nothing) lines (the identities) back to the double lines.

*Solution to Exercise 6.27.*

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  $\epsilon: By \rightarrow y$ : the on-position function is the unique map  $!: B \rightarrow 1$ , while every on-directions function  $1 \rightarrow 1$  must be the identity. Meanwhile  $By \triangleleft By \cong B^2y$ , so to specify a duplicator lens  $\delta: By \rightarrow B^2y$ , it suffices to specify an on-positions function  $\delta_1: B \rightarrow B^2$ , 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 map is just the canonical right projection, are both the identity on  $B$ . The only function  $\delta_1: B \rightarrow B \times B$  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 **(Set, 1, ×)**.)

*Solution to Exercise 6.33.*

1. The category  $\boxed{A \xrightarrow{f} B}$  has 2 morphisms out of  $A$ , namely  $\text{id}_A$  and  $f$ ; and 1 morphism out of  $B$ , namely  $\text{id}_B$ . So its carrier is  $\{A\}y^{\{\text{id}_A, f\}} + \{B\}y^{\{\text{id}_B\}} \cong y^2 + y$ .



2. The category  $\boxed{B \xrightarrow{g} A \xleftarrow{h} C}$  has 1 morphism out of  $A$ , namely  $\text{id}_A$ ; 2 morphisms out of  $B$ , namely  $\text{id}_B$  and  $g$ ; and 2 morphisms out of  $C$ , namely  $\text{id}_C$  and  $h$ . So its carrier is  $\{A\}y^{\{\text{id}_A\}} + \{B\}y^{\{\text{id}_B, g\}} + \{C\}y^{\{\text{id}_C, h\}} \cong 2y^2 + y$ .
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 6.40 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  $m \rightarrow n$  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  $n \rightarrow m$  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

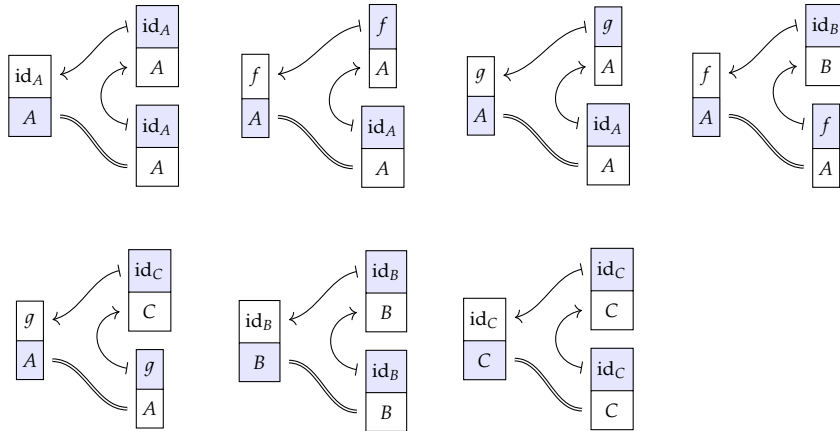
$$\sum_{n \in \mathbb{N}} y^{\{m \in \mathbb{N} \mid m \leq n\}} \cong \sum_{n \in \mathbb{N}} y^{n+1} \cong y^1 + y^2 + y^3 + \dots.$$

(This category is the poset  $(\mathbb{N}, \geq)$ .)

*Solution to Exercise 6.35.*

We are given a comonoid  $(\epsilon, \delta)$  corresponding to the preorder  $\boxed{B \xleftarrow{f} A \xrightarrow{g} C}$ .

1. There are three morphisms with domain  $A$ , namely  $\text{id}_A$ ,  $f$ , and  $g$ ; the only other morphisms are the identity morphisms on  $B$  and  $C$ . So the carrier is  $\epsilon = \{A\}y^{\{\text{id}_A, f, g\}} + \{B\}y^{\{\text{id}_B\}} + \{C\}y^{\{\text{id}_C\}}$ .
2. It suffices to specify the eraser  $\epsilon: \epsilon \rightarrow y$  on directions: as always,  $\epsilon_i^\# : 1 \rightarrow \epsilon[i]$  picks out  $\text{id}_i$  for each  $i \in \epsilon(1) = \{A, B, C\}$ .
3. The duplicator  $\delta: \epsilon \rightarrow \epsilon \triangleleft \epsilon$  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  $\delta$  using polyboxes as follows:



*Solution to Exercise 6.36.*

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 6.37.*

1. The polynomial  $p := y^{n+1} + ny$  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 6.39.*

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 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  $\sum_{n \in \mathbb{N}} |S|^n = |\mathbb{N}||S| = |S|$ . So the resulting category will again have carrier  $Sy^S$ .

*Solution to Exercise 6.41.*

The fact that monoids  $(M, e, *)$  in  $(\mathbf{Set}, 1, \times)$  are just comonoids  $(y^M, \epsilon, \delta)$  in  $(\mathbf{Poly}, y, \triangleleft)$ , following the construction of Example 6.40, is a direct consequence of the fully faithful Yoneda embedding  $\mathbf{Set}^{\text{op}} \rightarrow \mathbf{Poly}$  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 functions of  $\epsilon$  and  $\delta$ . The monoid’s unitality condition states that  $e$  is a 2-sided unit for  $*$ , or that

$$\begin{array}{ccccc} 1 \times M & \xlongequal{\quad} & M & \xlongequal{\quad} & M \times 1 \\ & \searrow e \times M & \uparrow * & \swarrow M \times e & \\ & & M \times M, & & \end{array}$$

commutes—equivalent to the comonoid’s erasure laws, which state that

$$\begin{array}{ccccc} y \triangleleft y^M & \xlongequal{\quad} & y^M & \xlongequal{\quad} & y^M \triangleleft y \\ & \swarrow \epsilon \triangleleft y^M & \downarrow \delta & \searrow y^M \triangleleft \epsilon & \\ & & y^M \triangleleft y^M, & & \end{array}$$

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

$$\begin{array}{ccc} M & \xleftarrow{\quad * \quad} & M \times M \\ \uparrow * & & \uparrow M \times * \\ M \times M & \xleftarrow{\quad * \times M \quad} & M \times M \times M, \end{array}$$

commutes—equivalent to the comonoid's coassociative law, which state that

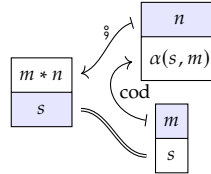
$$\begin{array}{ccc}
 y^M & \xrightarrow{\delta} & y^M \triangleleft y^M \\
 \delta \downarrow & & \downarrow y^M \triangleleft \delta \\
 y^M \triangleleft y^M & \xrightarrow{\delta \triangleleft y^M} & y^M \triangleleft y^M \triangleleft y^M,
 \end{array}$$

commutes (trivial on positions, equivalent to the monoid's associativity condition on directions).

*Solution to Exercise 6.44.*

Here  $(M, e, *)$  is a monoid,  $S$  is a set,  $\alpha: S \times M \rightarrow S$  is a monoid action, and  $\mathcal{ML}$  is the associated category, whose corresponding comonoid is  $(Sy^M, \epsilon, \delta)$ . We also know that for each  $s \in S$  and  $m \in M$ , there is a morphism  $s \xrightarrow{m} \alpha(s, m)$  in  $\mathcal{ML}$ .

1. The erasure  $\epsilon: Sy^M \rightarrow y$  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  $\epsilon$  be the lens whose on-directions function at each  $s \in S$  is  $\epsilon_s^\# : 1 \rightarrow M$  always maps to  $e$ .
2. The duplicator  $\delta: Sy^M \rightarrow Sy^M \triangleleft Sy^M$  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  $\delta$  in polyboxes as



3. We constructed  $\delta$  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  $\epsilon$  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  $\delta$ . 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  $\delta$  amounts to checking that composition plays nicely with codomains and is associative. We already checked the former when defining  $\delta$ , 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  $\mathcal{ML}$  is a category whose objects are the elements of the set  $S$  being acted on, and whose morphisms  $s \rightarrow t$  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 6.38 corresponds to a category in which every map is an isomorphism, so for each  $m \in M$ , the left-action  $\alpha(m, -): M \rightarrow M$  would need to be a bijection, and this is the case iff  $M$  is a group.

Solution to Exercise 6.46.

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 6.43. For example, the erasure  $(\mathbb{R}/\mathbb{Z})y^{\mathbb{R}} \rightarrow y$  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 6.48.

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 6.33 #4), the poset  $(\mathbb{N}, \leq)$  (see Exercise 6.33 #5), and the state category on  $\mathbb{N}$  (see Example 6.38).
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 6.52.

As in Definition 6.49, we have a monoidal category  $(\mathcal{C}, y, \triangleleft)$  with comonoids  $\mathcal{C} := (c, \epsilon, \delta)$  and  $\mathcal{C}' := (c', \epsilon', \delta')$  and a comonoid morphism  $F: \mathcal{C} \rightarrow \mathcal{C}'$  (really a morphism  $F: c \rightarrow c'$  in  $\mathcal{C}$ ). Let's throw in another comonoid  $\mathcal{C}'' := (c'', \epsilon'', \delta'')$  and another comonoid morphism  $G: \mathcal{C}' \rightarrow \mathcal{C}''$  (really a morphism  $G: c' \rightarrow c''$  in  $\mathcal{C}$ ).

1. To show that the identity morphism  $\text{id}_c: c \rightarrow c$  is a comonoid morphism  $\mathcal{C} \rightarrow \mathcal{C}$ , we must check that it preserves erasure by showing that (6.50) commutes, then check that it preserves duplication by showing that (6.50) commutes:

$$\begin{array}{ccc} c & \xrightarrow{\text{id}_c} & c \\ \epsilon \downarrow & & \downarrow \epsilon \\ y & \xlongequal{\quad} & y \end{array} \qquad \begin{array}{ccc} c & \xrightarrow{\text{id}_c} & c \\ \delta \downarrow & & \downarrow \delta \\ c \triangleleft c & \xrightarrow{\text{id}_c \triangleleft \text{id}_c} & c \triangleleft c. \end{array}$$

But they do commute, since  $\text{id}_c$  is the identity on  $c$  and  $\text{id}_c \triangleleft \text{id}_c$  is the identity on  $c \triangleleft c$ .

2. To show that the composite  $F \circ G: c \rightarrow c''$  of the two comonoid morphisms  $F$  and  $G$  is itself a comonoid morphism  $\mathcal{C} \rightarrow \mathcal{C}''$ , we check that it preserves erasure by showing that (6.50) commutes, then check that it preserves duplication by showing that (6.50) commutes:

$$\begin{array}{ccc} c & \xrightarrow{F \circ G} & c'' \\ \epsilon \downarrow & & \downarrow \epsilon'' \\ y & \xlongequal{\quad} & y \end{array} \qquad \begin{array}{ccc} c & \xrightarrow{F \circ G} & c'' \\ \delta \downarrow & & \downarrow \delta'' \\ c \triangleleft c & \xrightarrow{(F \circ G) \triangleleft (F \circ G)} & c'' \triangleleft c''. \end{array}$$

But we can rewrite these squares like so, using the fact that  $(F \circ G) \triangleleft (F \circ G) = (F \triangleleft F) \circ (G \triangleleft G)$  on the right:

$$\begin{array}{ccc} c & \xrightarrow{F} & c & \xrightarrow{G} & c'' \\ \epsilon \downarrow & & \epsilon' \downarrow & & \downarrow \epsilon'' \\ y & \xlongequal{\quad} & y & \xlongequal{\quad} & y \end{array} \qquad \begin{array}{ccccc} c & \xrightarrow{F} & c' & \xrightarrow{G} & c'' \\ \delta \downarrow & & \delta' \downarrow & & \downarrow \delta'' \\ c \triangleleft c & \xrightarrow{F \triangleleft F} & c' \triangleleft c' & \xrightarrow{G \triangleleft G} & c'' \triangleleft c''. \end{array}$$

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 6.59.

Here  $F: \mathcal{C} \rightarrow \mathcal{D}$  and  $G: \mathcal{D} \rightarrow \mathcal{E}$  are cofunctors in  $\mathbf{Cat}^\#$ .

1. The identity cofunctor  $\text{id}_{\mathcal{D}}$  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  $\text{id}_{\mathcal{D}}$  sends each object  $d \in \text{Ob } \mathcal{D}$  to itself, while  $(\text{id}_{\mathcal{D}})^\#_d: \mathcal{D}[d] \rightarrow \mathcal{D}[d]$  sends each morphism out of  $d$  to itself as well.
2. The composite cofunctor  $F \circ G: \mathcal{C} \rightarrow \mathcal{E}$  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,  $F \circ G$  sends each  $c \in \text{Ob } \mathcal{C}$  to  $G(Fc) \in \text{Ob } \mathcal{E}$ . Then given  $c \in \text{Ob } \mathcal{C}$ , the on-morphisms function  $(F \circ G)_c^\#: \mathcal{E}[G(Fc)] \rightarrow \mathcal{C}[c]$  is the composite of on-directions functions

$$\mathcal{E}[G(Fc)] \xrightarrow{G_{Fc}^\#} \mathcal{D}[Fc] \xrightarrow{F_c^\#} \mathcal{C}[c],$$

sending each morphism  $h$  with domain  $G(Fc)$  to  $F_c^\#(G_{Fc}^\#h)$ .

Solution to Exercise 6.61.

We want to show that categories  $\mathcal{C}$  and  $\mathcal{D}$  are isomorphic in  $\mathbf{Cat}$  if and only if they are isomorphic in  $\mathbf{Cat}^\#$ . First, assume that  $\mathcal{C}$  and  $\mathcal{D}$  are isomorphic in  $\mathbf{Cat}$ , so that there exist mutually inverse functors  $F: \mathcal{C} \rightarrow \mathcal{D}$  and  $G: \mathcal{D} \rightarrow \mathcal{C}$ . Then we can define a cofunctor  $H: \mathcal{C} \rightarrow \mathcal{D}$  such that for each  $c \in \mathcal{C}$  we have  $Hc := Fc \in \mathcal{D}$ , and for each  $g \in \mathcal{D}[Fc]$  we have  $H_c^\#g := Gg \in \mathcal{C}[GFc] = \mathcal{C}[c]$ . We can verify that  $H$  really is a cofunctor: it preserves identities and composition because  $G$  does, and it preserves codomains because  $H \text{ cod } H_c^\#g = F \text{ cod } Gg = FG \text{ cod } g = \text{cod } g$ . Analogously, we can define a cofunctor  $K: \mathcal{D} \rightarrow \mathcal{C}$  with  $Kd := Gd$  and  $K_d^\#f := Ff$  for each  $d \in \mathcal{D}$  and  $f \in \mathcal{C}[Gd]$ . Then  $H \circ K$  is equal to  $F \circ G$  both on objects and on morphisms, so it is the identity cofunctor on  $\mathcal{C}$ ; analogously,  $K \circ H$  is the identity cofunctor on  $\mathcal{D}$ . Thus  $\mathcal{C}$  and  $\mathcal{D}$  are isomorphic in  $\mathbf{Cat}^\#$ .

Conversely, assume  $\mathcal{C}$  and  $\mathcal{D}$  are isomorphic in  $\mathbf{Cat}^\#$ , so that there exist mutually inverse cofunctors  $H: \mathcal{C} \rightarrow \mathcal{D}$  and  $K: \mathcal{D} \rightarrow \mathcal{C}$ . Given objects  $c, c'$  and a morphism  $f: c \rightarrow c'$  in  $\mathcal{C}$ , we have that  $KHc = c$ , so  $K_{Hc}^\#$  is a function  $\mathcal{C}[c] \rightarrow \mathcal{D}[Hc]$ . In particular,  $K_{Hc}^\#f$  is a morphism in  $\mathcal{D}$  whose domain is  $Hc$  and whose codomain satisfies  $K \text{ cod } K_{Hc}^\#f = \text{cod } f = c'$ , and thus  $\text{cod } K_{Hc}^\#f = Hc'$ . Hence we can define a functor  $F: \mathcal{C} \rightarrow \mathcal{D}$  such that for each  $c \in \mathcal{C}$  we have  $Fc := Hc \in \mathcal{D}$ , and for each morphism  $f: c \rightarrow c'$  in  $\mathcal{C}$  we have  $Ff := K_{Hc}^\#f: Hc \rightarrow Hc'$ . Functoriality follows from the fact that  $K$  preserves identities and composition. Analogously, we can define a functor  $G: \mathcal{D} \rightarrow \mathcal{C}$  with  $Gd := Kd$  for each  $d \in \mathcal{D}$  and  $Gg := H_{Kd}^\#g$  for each  $g: d \rightarrow d'$  in  $\mathcal{D}$ . Then  $F \circ G$  is equal to  $H \circ K$  on objects; on morphisms, it sends each  $f: c \rightarrow c'$  in  $\mathcal{C}$  to  $H_{KHc}^\#(K_{Hc}^\#f) = H_c^\#(K_{Hc}^\#f) = (K \circ H)_c^\#f = f$  itself. So  $F \circ G$  is the identity functor on  $\mathcal{C}$ . Analogously,  $G \circ F$  is the identity functor on  $\mathcal{D}$ . Thus  $\mathcal{C}$  and  $\mathcal{D}$  are isomorphic in  $\mathbf{Cat}$ .

Solution to Exercise 6.63.

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 6.39, for the case of  $S := 1$ .
2. For any category  $\mathcal{C}$ , there is a unique cofunctor  $\mathcal{C} \rightarrow y$ : 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 6.62, a cofunctor 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 6.65.

Given a cofunctor  $F: \mathcal{C} \rightarrow \mathcal{A}$ , where  $\mathcal{A}$  is the walking arrow category as in Example 6.64, assume  $Fc = s$  for all  $c \in \mathcal{C}$ . By Example 6.64, 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  $a: s \rightarrow t$  in  $\mathcal{A}$  to be sent back to via  $F^\#$ . But there are no objects in  $\mathcal{C}$

that  $F$  sends to  $t$ . So the only way such a cofunctor 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 6.66.*

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 cofunctor  $\mathcal{C} \rightarrow y^{n+1} + ny$  comprises an assignment of a label to each object in  $\mathcal{C}$ : either it assigns “center” or it assigns an element of  $\{1, 2, \dots, n\}$ . If it assigns “center”, the object is equipped with  $n$ -many morphisms, with the  $i$ th one having as its codomain an object labeled  $i$ .
2. A cofunctor  $\mathcal{C} \rightarrow (\mathbb{N}, \leq)$  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 cofunctor  $\mathcal{C} \rightarrow (\mathbb{N}, \leq)$  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 6.68.*

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 cofunctors are

$$\begin{array}{ccccccc}
 t & t & s \rightarrow t & s & t & s & t \\
 & & & \downarrow & & \searrow & \downarrow \\
 t & t & t & t & t & t & s \rightarrow t \\
 s & s & s & s & s \rightarrow t & s & t \\
 \downarrow & \downarrow & \searrow & \downarrow & \searrow & \downarrow & \searrow \\
 t & t & t & t & s \rightarrow t & s \rightarrow t & s \rightarrow t
 \end{array}$$

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 cofunctors are

$$z \quad z \quad x \rightarrow z \quad y \rightarrow z$$

*Solution to Exercise 6.69.*

1. Let  $\mathcal{P}$  be a poset. A cofunctor  $y \rightarrow \mathcal{P}$  represents an maximal element  $p \in \mathcal{P}$ . Indeed, if there were some  $p \leq p'$  with  $p' \neq p$ , then the cofunctor would have to send the map  $p \rightarrow p'$  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  $m = \boxed{\bullet^0 \rightarrow \dots \rightarrow \bullet^{m-2} \rightarrow \bullet^{m-1}}$ . By the same reasoning as above, a map  $[m] \rightarrow [n]$  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 cofunctors. For example, the cofunctors  $[2] \rightarrow [2]$  are those labeled by  $(2, 2, 2)$ ,  $(1, 2, 2)$ ,  $(1, 1, 2)$ , and  $(0, 1, 2)$ .

*Solution to Exercise 6.72.*

We seek the number of arrow fields on the category  $\boxed{\bullet \rightarrow \bullet}$ . 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 6.73.*

1. A cofunctor  $\mathcal{C} \rightarrow y^{\mathbb{Z}}$  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  $\text{cod}(c.1) = c'$ .

2. We seek a canonical cofunctor  $y^{\mathbb{Z}} \rightarrow y^{\mathbb{N}}$ . The canonical inclusion  $i: \mathbb{N} \hookrightarrow \mathbb{Z}$  gives rise to a lens  $\iota$  from  $y^{\mathbb{Z}}$  to  $y^{\mathbb{N}}$ , whose sole on-directions function  $\iota^{\sharp}: \mathbb{N} \hookrightarrow \mathbb{Z}$  coincides with  $i$ . We verify that  $\iota$  is a cofunctor: 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 6.74.*

1. Cofunctors  $y^M \rightarrow y^N$  are the same as monoid homomorphisms  $N \rightarrow M$ . See Proposition 6.78.
2. No! This is the weirdest thing about cofunctors. For example, there is a unique cofunctor  $y \rightarrow y^2 + y$  from the walking object to the walking arrow, so if we reverse the arrows, there is no longer a cofunctor that acts the same on objects.

*Solution to Exercise 6.75.*

We are given a monoid  $(M, e, *)$ , a set  $S$ , and an  $M$ -action  $\alpha: M \times S \rightarrow S$ . 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  $Sy^M \rightarrow y^M$  sends the identity back to the identity, trivially preserves codomains, and also preserves composition, so it is a cofunctor.

*Solution to Exercise 6.77.*

This works!

*Solution to Exercise 6.80.*

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 6.85.*

Given a bijection of sets  $S \cong T \times U$ , we seek a unique cofunctor  $Sy^S \rightarrow Ty^T$  whose on-positions function  $\text{get}$  is given by the product projection  $S \cong T \times U \rightarrow T$ . Such a cofunctor should also have an on-directions function  $\text{put}: S \times T \rightarrow S$  such that the three laws from Example 6.84 are satisfied. With  $S \cong T \times U$ , such a function is uniquely determined by its components

$$S \times T \xrightarrow{\text{put}} S \xrightarrow{\text{get}} T \quad \text{and} \quad S \times T \xrightarrow{\text{put}} S \xrightarrow{\pi} U,$$

where  $\pi: S \cong T \times U \rightarrow U$  is the other product projection. The left component is determined by the put-get law, which specifies the behavior of the composite  $\text{put} \circ \text{get}$ : 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  $\text{get}$  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  $\pi$  to both sides yields

$$\pi(\text{put}((t, u), t)) = u \quad \text{and} \quad \pi(\text{put}(\text{put}((t, u), t'), t'')) = \pi(\text{put}((t, u), t''))$$

*Solution to Exercise 6.86.*

1. There are 6 product projections  $3 \rightarrow 3$ , namely the three automorphisms, so the answer is 6.
2. There are 6 product projections  $4 \rightarrow 2$ , so the answer is 6.



*Solution to Exercise 6.88.*

Yes, it is a cofunctor. One sees easily that it sends identities back to identities and composites back to composites. The codomain of  $f \circ g$  is simply  $f \circ g$  as an element of  $\mathfrak{c}[i]$ , and it is sent forward to  $\text{cod}(f \circ g)$ , so the map preserves codomains.

*Solution to Exercise 6.90.*

1. Every object in  $\mathcal{U}$  must be labeled with either 1 or 2, but there are no other requirements.
2. If cofunctors  $\mathcal{U} \rightarrow \mathcal{C}$  are always in bijection with objects in  $\mathcal{C}$ , then with  $\mathcal{C} = 2y$ , we have a bijection  $2 \cong 2^{\text{Ob}(\mathcal{U})}$  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 cofunctor, whereas there are two objects in  $\mathcal{D}$ . We conclude that objects are not representable in  $\mathbf{Cat}^\#$  the way they are in  $\mathbf{Cat}$ .
4. No, there is no such  $\mathcal{V}$ . There is exactly one cofunctor  $0 \rightarrow \mathcal{V}$ , but there are no objects of 0.

*Solution to Exercise 6.92.*

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 6.93.*

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  $\text{id}_c^{\text{old}}$  is no longer an identity because  $\text{id}_c^{\text{old}} \circ i = \text{id}_c^{\text{old}} \neq i$ .
2. Given a cofunctor  $\varphi: \mathfrak{c} \rightarrow \mathfrak{d}$ , we get a cofunctor  $\mathfrak{c}y \rightarrow \mathfrak{d}y$  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 cofunctor.
3. It is a monad: there is an obvious unit map  $\mathfrak{c} \rightarrow \mathfrak{c}y$  and a multiplication map  $\mathfrak{c}yy \rightarrow \mathfrak{c}y$  that sends the new identity back to the newest identity.

*Solution to Exercise 6.94.*

1. Counterexample: take  $\mathfrak{c} = 0$ . Then the unique map  $f: \mathfrak{c} \rightarrow \mathfrak{d}$  is a cofunctor and so is  $f \circ g$  for any lens  $g: \mathfrak{d} \rightarrow \mathfrak{e}$ , but some lenses are not cofunctors.
2. Counterexample: take  $\mathfrak{e} = y$ . Then there is a unique cofunctor  $\mathfrak{e}: \mathfrak{c} \rightarrow y$ . As long as  $f$  is copointed, meaning it sends identities backward to identities, then  $f \circ g$  will be a cofunctor. But not every copointed lens is a cofunctor.

*Solution to Exercise 6.100.*

In general, a functor  $F: \mathcal{C} \rightarrow \mathcal{D}$  is a discrete opfibration iff, for every object  $c \in \mathcal{C}$ , the induced map  $F[c]: \mathcal{C}[c] \rightarrow \mathcal{D}[Fc]$  is a bijection.

In our case, if  $\pi$  and  $\pi'$  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 6.101.*

1. The identity map  $\text{id}_i$  satisfies  $\pi(\text{id}_i) = \text{id}_{\pi(i)}$ , so  $\overline{\text{id}_{\pi(i)}} = \text{id}_i$  by the uniqueness-of-lift condition in Definition 6.98.
2. The morphism  $\bar{f} \circ \bar{g}$  satisfies  $\pi(\bar{f} \circ \bar{g}) = \pi(\bar{f}) \circ \pi(\bar{g}) = f \circ g$ , so again this follows by uniqueness of lift.
3. To see that  $\pi$  is a cofunctor, 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  $\pi$  as a functor. Given an object  $s \in \mathcal{S}$ , the action of  $\pi$  backward on morphisms is the lift operation



$\pi_s^\sharp(f) := \bar{f}$ . This preserves identity by part 1, composition by part 2, and codomains because  $\pi(\text{cod}(\pi_s^\sharp(f))) = \text{cod}(\pi(\pi_s^\sharp(f))) = \text{cod}(f)$ .

*Solution to Exercise 6.104.*

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:  $\pi(\text{id}) = \text{id}$  and  $\pi(f \circ g)$  is  $f \circ g$ .
2. Given an object  $(c, x) \in \int^{\mathcal{C}} I$  and  $f: c \rightarrow c'$ , we need to check that there is a unique object  $s \in \int^{\mathcal{C}} I$  and morphism  $\bar{f}: (c, x) \rightarrow s$  with  $\pi(\bar{f}) = f$ . Using  $x' := I(f)(x)$  and  $s := (c', x')$  and  $\bar{f} := f$ , we do indeed get  $\pi(\bar{f}) = f$  and we find that it is the only possible choice.

*Solution to Exercise 6.105.*

1. Given a natural transformation  $\alpha: I \rightarrow J$ , we need a functor  $\int^{\mathcal{C}} \alpha: \int^{\mathcal{C}} I \rightarrow \int^{\mathcal{C}} J$ . 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 6.106.*

Given a graph  $G := (\text{src}, \text{tgt}: A \rightrightarrows V)$  and functor  $S: \mathcal{G} \rightarrow \mathbf{Set}$ , define a new graph  $H$  as the following diagram of sets:

$$\sum_{a \in A} S(\text{src}(a)) \xrightleftharpoons[(\text{tgt}, S(a))]{(\text{src}, \text{id})} \sum_{v \in V} S(v)$$

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  $a: v \rightarrow v'$ , 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  $S(a): S(v) \rightarrow S(v')$  is the function given by the functor  $S: \mathcal{G} \rightarrow \mathbf{Set}$ .

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  $f: v \rightarrow v'$  in  $\mathcal{G}$ , but this is just a length- $n$  sequence  $(a_1, \dots, a_n)$  of arrows, with  $v = \text{src}(a_1)$ ,  $\text{tgt}(a_i) = \text{src}(a_{i+1})$  for  $1 \leq i < n$ , and  $\text{tgt}(a_n) = v'$ . This is the same data as a morphism in the free category on  $H$ .

*Solution to Exercise 6.112.*

\*\*

*Solution to Exercise 6.113.*

\*\*



# Categorical properties of polynomial comonoids

While we defined the category  $\mathbf{Cat}^\sharp$  of categories and cofunctors in the last chapter, we have only begun to scratch the surface of the properties it satisfies. As the category of comonoids in  $\mathbf{Poly}$ , equipped with a canonical forgetful functor  $U: \mathbf{Cat}^\sharp \rightarrow \mathbf{Poly}$  sending each polynomial to its carrier,  $\mathbf{Cat}^\sharp$  inherits many of the categorical properties satisfied by  $\mathbf{Poly}$ . Underpinning this inheritance is the fact that  $U$  has a right adjoint, a cofree functor  $\mathbf{Poly} \rightarrow \mathbf{Cat}^\sharp$ . We will introduce this adjunction in the first section of this chapter. Then we will discuss how many of the categorical properties of  $\mathbf{Poly}$ , including much of what we covered in Chapter 4, play nicely with restriction to  $\mathbf{Cat}^\sharp$ . Finally, we will touch on other constructions we can make over the comonoids in  $\mathbf{Poly}$ , such as their comodules and coalgebras.

## 7.1 Cofree comonoids

Consider a dynamical system  $\varphi: s \rightarrow p$  with state system  $s$  and interface  $p$ . In Section 6.1.5, we posed the question of whether there was a single morphism that could capture all the information encoded by the family of lenses  $\text{Run}_n(\varphi): s \rightarrow p^{\triangleleft n}$  for all  $n \in \mathbb{N}$ , defined as the composite

$$s \xrightarrow{\delta^{(n)}} s^{\triangleleft n} \xrightarrow{\varphi^{\triangleleft n}} p^{\triangleleft n}$$

that models  $n$  runs through the system  $\varphi$ .

It turns out that there is: the key is that there is a natural way to interpret every polynomial  $p$  as a category  $\mathcal{T}_p$ , so that cofunctors into  $\mathcal{T}_p$  are exactly lenses into  $p$ . In other words,  $\mathcal{T}_p$  will turn out to be the *cofree comonoid* (or *cofree category*) on  $p$ . Cofree comonoids in  $\mathbf{Poly}$  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  $U: \mathbf{Cat}^\# \rightarrow \mathbf{Poly}$  has a right adjoint  $\mathcal{T}_-: \mathbf{Poly} \rightarrow \mathbf{Cat}^\#$ , where for each  $p \in \mathbf{Poly}$ , the carrier  $t_p := U\mathcal{T}_p$  of the category  $\mathcal{T}_p$  is given by the limit of the following diagram:

$$\begin{array}{ccccccc}
 y & & p & & p^{\triangleleft 2} & & p^{\triangleleft 3} & & \dots \\
 \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\
 1 & \xleftarrow{!} & p \triangleleft 1 & \xleftarrow{p \triangleleft !} & p^{\triangleleft 2} \triangleleft 1 & \xleftarrow{p^{\triangleleft 2} \triangleleft !} & p^{\triangleleft 3} \triangleleft 1 & \xleftarrow{\dots} & \dots
 \end{array} \tag{7.1}$$

Thus, we will show that  $U$  and  $\mathcal{T}_-$  form a forgetful-cofree adjunction, making  $\mathcal{T}_p$  the cofree comonoid on  $p$ . But first, let us concretely characterize the canonical comonoid structure on the limit of (7.1), before showing that it is indeed cofree.

### 7.1.1 The carrier of the cofree comonoid

Let  $t_p$  be the limit of the diagram (7.1) in  $\mathbf{Poly}$ ; 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 7.2** (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  $\text{tree}_p$ .

We can think of a  $p$ -tree as being “built” out of  $p$ -corollas according to these instructions:

To choose a  $p$ -tree in  $\text{tree}_p$ :

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  $\text{tree}_p$ :

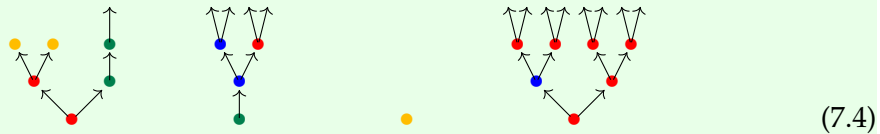
1. choose a  $p$ -corolla:
  - its root  $i_0 \in p(1)$  will be (the label of) the tree's root, and
  - its leaves in  $p[i_0]$  will be (the labels of) the edges out of the root;
2. for each  $p[i_0]$ -leaf  $a_1$ :
  - 2.1. choose a  $p$ -tree in  $\text{tree}_p$ :
    - it will be the subtree whose 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 7.3* (A few example  $p$ -pretrees). Let  $p := \{\bullet, \circ\}y^2 + \{\circ\}y + \{\bullet\}$ . Here are four partially constructed  $p$ -trees:



Here only the third one—the single yellow dot—would count as an element of  $\text{tree}_p$ . After all, in Definition 7.2, 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 (7.4) have leaves emanating from the top

that have not been filled by any  $p$ -corollas, these trees are not elements of  $\text{tree}_p$ .

However, each of these trees can be extended to an actual element of  $\text{tree}_p$  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 7.5.* Let  $q := y^2 + 3y^1$ . 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 (7.4) 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 5.1.3 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 7.6.** 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 7.7.* In Definition 7.6, 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 (7.4) 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 7.8* (Trimming pretrees). Since  $p^{\triangleleft 1}(1) \cong p(1)$  and  $p^{\triangleleft 0}(1) \cong y(1) \cong 1$ , the unique function  $! : p(1) \rightarrow 1$  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  $\mathbf{Poly}$  of the identity on  $p^{\triangleleft n}$  and  $!$ , interpreted as a lens between constant polynomials, to obtain a lens  $p^{\triangleleft n} \triangleleft ! : p^{\triangleleft n} \triangleleft p(1) \rightarrow p^{\triangleleft n} \triangleleft 1$ , or equivalently a function  $p^{\triangleleft n}(!) : p^{\triangleleft n+1}(1) \rightarrow p^{\triangleleft n}(1)$  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  $p^{<(n+1)}$ -pretree’s height- $(n + 1)$  leaves, along with all the position-labels on its height- $n$  vertices. Thus  $p^{<n}(!)$  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

$$p^{<n}(1) \xleftarrow{p^{<n}(!)} p^{<(n+1)}(1) \leftarrow \dots \leftarrow p^{<(n+k-1)}(1) \xleftarrow{p^{<(n+k-1)}(!)} p^{<(n+k)}(1) \quad (7.9)$$

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 7.10* (A few more actual  $p$ -trees). Keeping  $p := \{\bullet, \bullet\}y^2 + \{\bullet\}y + \{\bullet\}$ , here are some elements of  $\text{tree}_p$  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.<sup>1</sup>
- 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  $\text{tree}_p$  (even just  $\text{tree}_{2y}$  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  $\text{tree}_p$  cannot even be described.

*Exercise 7.11.* For each of the following polynomials  $p$ , characterize the set of trees  $\text{tree}_p$ .

5.  $p := 1$ .
6.  $p := 2$ .
7.  $p := y$ .
8.  $p := y^2$ .
9.  $p := 2y$ .

<sup>1</sup>To formalize the notions of “left” and “right,” we could think of the direction-sets of the red and blue dots as  $2 \cong \{\text{left}, \text{right}\}$ , so that out of every vertex there is an edge labeled left and an edge labeled right.

10.  $p := y + 1$ .
11.  $p := By^A$  for some sets  $A, B \in \mathbf{Set}$ . ◇

**Exercise 7.12.** 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  $\mathbf{tree}_p$  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  $\mathbf{tree}_q$  is the set of  $L$ -labeled  $n$ -ary trees? ◇

From here, a natural question to ask is how the set  $\mathbf{tree}_p$  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 7.6 gives a clue: every  $p$ -tree has a stage- $n$  pretree obtained by removing all vertices of height greater than  $n$ , yielding a function  $\mathbf{tree}_p \rightarrow p^{\triangleleft n}(1)$  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  $p^{\triangleleft(n+k)}(1) \rightarrow p^{\triangleleft n}(1)$  from (7.9) in Example 7.8. In particular, the following diagram commutes for all  $n \in \mathbb{N}$ :

$$\begin{array}{ccc} \mathbf{tree}_p & & \\ \pi^{(n)} \downarrow & \searrow \pi^{(n+1)} & \\ p^{\triangleleft n}(1) & \xleftarrow{p^{\triangleleft n}(!)} & p^{\triangleleft(n+1)}(1). \end{array}$$

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 7.8:

$$\begin{array}{ccccccc} \mathbf{tree}_p & & & & & & \\ \pi^{(0)} \downarrow & \searrow \pi^{(1)} & \searrow \pi^{(2)} & \searrow \pi^{(3)} & \searrow \dots & & \\ 1 & \xleftarrow{!} & p(1) & \xleftarrow{p(!)} & p^{\triangleleft 2}(1) & \xleftarrow{p^{\triangleleft 2}(!)} & p^{\triangleleft 3}(1) \xleftarrow{\dots} \dots \end{array} \quad (7.13)$$

So  $\mathbf{tree}_p$  with the functions  $\pi^{(n)}$  forms a cone over 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 7.14.* Prove that  $\text{tree}_p$  with the functions  $\pi^{(n)}: \text{tree}_p \rightarrow p^{\triangleleft n}(1)$  is the limit of the diagram

$$1 \xleftarrow{!} p(1) \xleftarrow{p(!)} p^{\triangleleft 2}(1) \xleftarrow{p^{\triangleleft 2}(!)} p^{\triangleleft 3}(1) \xleftarrow{\quad} \cdots \quad (7.15)$$

You may use the fact that (7.13) commutes.  $\diamond$

If you know about coalgebras for functors, as mentioned in Example 5.73, then you might know the limit  $t_p(1)$  of (7.15) 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 7.16.* A  $p$ -coalgebra morphism between  $p$ -coalgebras  $\varphi: S \rightarrow p(S)$  and  $\psi: T \rightarrow p(T)$  (as in Example 5.73) is a function  $f: S \rightarrow T$  such that the square

$$\begin{array}{ccc} S & \xrightarrow{\varphi} & p(S) \\ f \downarrow & & \downarrow p(f) \\ T & \xrightarrow{\psi} & p(T) \end{array}$$

commutes.

1. Choose what you think is a good function  $\text{tree}_p \rightarrow p(\text{tree}_p)$ .
2. Show that  $\text{tree}_p$  equipped with your function  $\text{tree}_p \rightarrow p(\text{tree}_p)$  is the terminal object in the category of  $p$ -coalgebras and the morphisms between them.
3. Show that the function  $\text{tree}_p \rightarrow p(\text{tree}_p)$  you chose is a bijection.

$\diamond$

### Positions of the cofree comonoid

Now recall from Example 4.34 that in **Poly**, the positions of a limit are the limit of the positions. Moreover, in (7.1), every vertical map  $p^{\triangleleft n} \rightarrow p^{\triangleleft n} \triangleleft 1$  is, in fact, a *vertical lens* in the sense of Definition 4.50: an isomorphism on positions. So (7.1) on positions collapses down to its bottom row, viewed as a diagram in **Set**. Yet this is precisely the diagram from (7.15), whose limit is  $\text{tree}_p$ . So  $\text{tree}_p$  is the position-set of the limit  $t_p$  of (7.1): we write  $t_p(1) \cong \text{tree}_p$ .

So, as we said above,  $t_p(1)$  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  $t_p$  is the limit of (7.1), Example 4.34 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 (7.1) has an empty direction-set, and there are no arrows between polynomials in the top row. So the directions of  $t_p$  are given by a coproduct of the directions of each  $p^{\ast n}$ .

More precisely, given a  $p$ -tree  $T \in \text{tree}_p$  whose stage- $n$  pretree for  $n \in \mathbb{N}$  is  $\pi^{(n)}T$ , the direction-set  $t_p[t]$  is given by the following coproduct:

$$t_p[T] := \sum_{n \in \mathbb{N}} p^{\ast n}[\pi^{(n)}T].$$

But by Definition 7.6, each  $p^{\ast n}[\pi^{(n)}T]$  is the set of height- $n$  leaves of the  $p^{\ast n}$ -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  $p^{\ast n}[\pi^{(n)}T]$  with the set of height- $n$  vertices of  $T$ ; we denote this set by  $\text{vtx}_n(T)$ .

Since the coproduct above ranges over all  $n \in \mathbb{N}$ , it follows that  $t_p[T]$  is the set of *all* vertices in  $T$ ; we denote this set by  $\text{vtx}(T)$ . The on-directions function at  $T$  of each projection  $t_p \rightarrow p^{\ast n}$  from the limit must then be the canonical inclusion

$$p^{\ast n}[\pi^{(n)}T] \cong \text{vtx}_n(T) \hookrightarrow \text{vtx}(T) \cong t_p[T],$$

sending height- $n$  leaves of  $\pi^{(n)}T$  to height- $n$  vertices of  $T$ .

Here is alternative way to think about the directions of  $t_p$  and each  $p^{\ast n}$  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  $p^{\ast n}$  at a given  $p^{\ast n}$ -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  $T \in \text{tree}_p$ , the direction-set  $t_p[T]$  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  $t_p$ . When we wish to refer to what one might call an infinite (rooted) “path,” we will instead call it a (rooted) *ray*.

*Exercise 7.17.* For each of the following polynomials  $p$ , characterize the polynomial  $t_p$ . You may choose to think of the directions of  $t_p$  either as vertices or as rooted paths. (Note that these are the same polynomials from Exercise 7.11.)

1.  $p := 1$ .
2.  $p := 2$ .
3.  $p := y$ .
4.  $p := y^2$ .
5.  $p := 2y$ .
6.  $p := y + 1$ .

7.  $p := By^A$  for some sets  $A, B \in \mathbf{Set}$ . ◇

We summarize the results of this section in the following proposition, thus concretely characterizing the carrier  $t_p$  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 (7.1) by  $\epsilon_p^{(n)}: t_p \rightarrow p^{\triangleleft n}$ . We denote  $\epsilon$  simply by  $\epsilon$ .

**Proposition 7.18.** For  $p \in \mathbf{Poly}$ , let

$$t_p := \sum_{T \in \mathbf{tree}_p} y^{\mathbf{vtx}(T)}$$

be the polynomial whose positions are  $p$ -trees and whose directions at each  $p$ -tree are the rooted paths. Then  $t_p$  is the limit of the diagram (7.1), with projections  $\epsilon_p^{(n)}: t_p \rightarrow p^{\triangleleft n}$  for every  $n \in \mathbb{N}$  making the following diagram commute:

$$\begin{array}{ccccccc}
 t_p & & & & & & \\
 \downarrow \epsilon_p & \searrow \epsilon_p^{(1)} & \searrow \epsilon_p^{(2)} & \searrow \epsilon_p^{(3)} & \searrow \dots & & \\
 y & p & p^{\triangleleft 2} & p^{\triangleleft 3} & \dots & & \\
 \downarrow & \downarrow & \downarrow & \downarrow & & & \\
 1 & p(1) & p^{\triangleleft 2}(1) & p^{\triangleleft 3}(1) & \dots & & \\
 & \longleftarrow ! & \longleftarrow p(!) & \longleftarrow p^{\triangleleft 2}(!) & \longleftarrow p^{\triangleleft 3}(!) & \longleftarrow \dots & 
 \end{array}
 \tag{7.19}$$

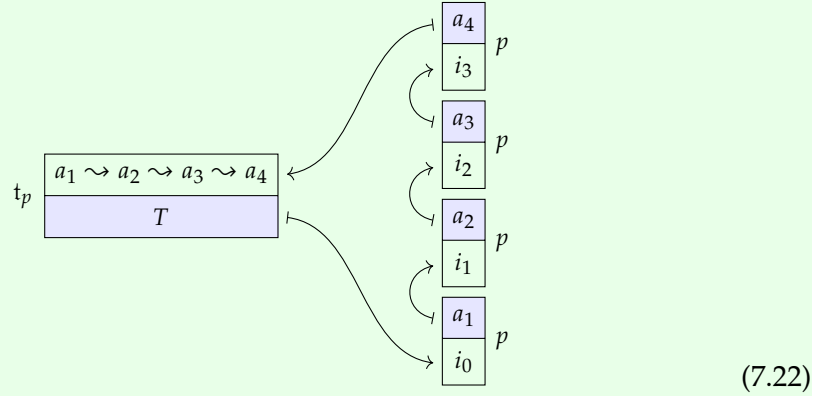
The lens  $\epsilon_p^{(n)}: t_p \rightarrow p^{\triangleleft n}$  sends each  $p$ -tree  $T \in \mathbf{tree}_p$  to its stage- $n$  pretree  $\pi^{(n)}T$  on positions and each height- $n$  leaf of  $\pi^{(n)}T$  to the corresponding height- $n$  rooted path of  $T$  on directions.

*Example 7.20* (Drawing  $\epsilon_p^{(n)}$  in polyboxes). Here is  $\epsilon_p^{(n)}: t_p \rightarrow p^{\triangleleft n}$  drawn in polyboxes, where we continue to denote the stage- $n$  pretree of a  $p$ -tree by  $\pi^{(n)}T$ :

$$\begin{array}{ccc}
 \mathbf{vtx}(-) & \boxed{v} & \longleftrightarrow & \boxed{v} & \mathbf{vtx}_n(-) \\
 \mathbf{tree}_p & \boxed{T} & \longmapsto & \boxed{\pi^{(n)}T} & p^{\triangleleft n}(1) \\
 & t_p & & p^{\triangleleft n} & 
 \end{array}
 \tag{7.21}$$

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  $T$ .

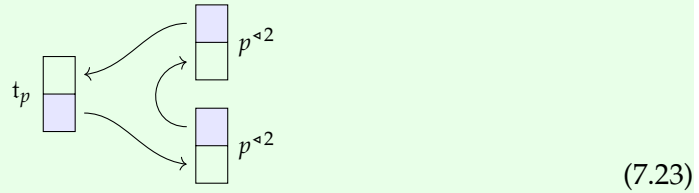
This isn't the only way we can write this lens in polyboxes, however; polyboxes have special notation for lenses to composites, allowing us to write, say, the  $n = 4$  case like so:



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.

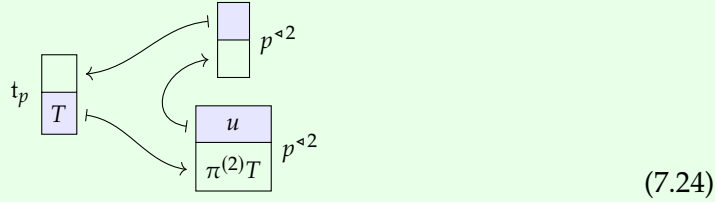
The contents of the position-boxes and the arrows going up on the codomain side carry all the data of the bottom 4 levels of  $T$ : namely the label  $i_0$  of the root, the label  $i_1$  of the vertex at the end of every direction  $a_1$  out of the root, and so on until  $i_3$ . All this specifies 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 (7.22) when their blue boxes are yet to be filled. So the position and direction depicted on the codomain side of (7.22) is equivalent to the position and direction depicted on the codomain side of (7.21). 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  $\epsilon_p^{(4)} : t_p \rightarrow p^{\triangleleft 4}$  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:

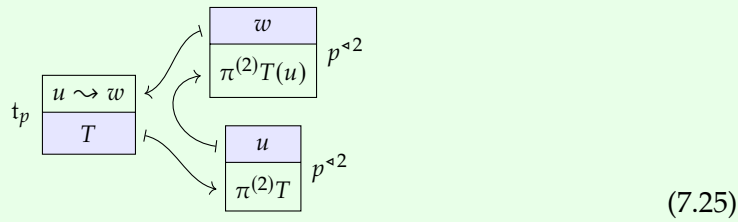


Let's think about how we should fill these boxes. We can still put a  $p$ -tree  $T$  in the lower left position box. From the commutativity of (7.19), we know the on-positions function  $\pi^{(4)}$  of  $\epsilon_p^{(4)}$  factors through the on-positions function  $\pi^{(2)}$  of  $\epsilon_p^{(2)}$ , 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  $p^{\aleph^2}$  on the right side of (7.23) are equivalent to the polyboxes for the 2 lower copies of  $p$  on the right side of (7.22)—just like how the polyboxes for  $p^{\aleph^n}$  in (7.21) are equivalent to the polyboxes for all  $n = 4$  copies of  $p$  in (7.22). There, the position could be represented with a single  $p^{\aleph^n}$ -pretree  $\pi^{(n)}T$ , and the direction  $(a_1, \dots, a_n)$  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  $p^{\aleph^2}$ , if we let  $u$  be the height-2 vertex of  $T$  at the end of the rooted path  $(a_1, a_2)$ :



But then the polyboxes for the 2 upper copies of  $p$  from (7.22) should also collapse down to a single pair of polyboxes for  $p^{\aleph^2}$ , with a  $p^{\aleph^2}$ -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  $p^{\aleph^2}$  in (7.24), we are starting over from  $u$  and moving up the subtree of  $T$  rooted at  $u$ —or, more precisely, the  $p^{\aleph^2}$ -pretree of the subtree rooted at  $u$ . So the polyboxes end up looking like this:

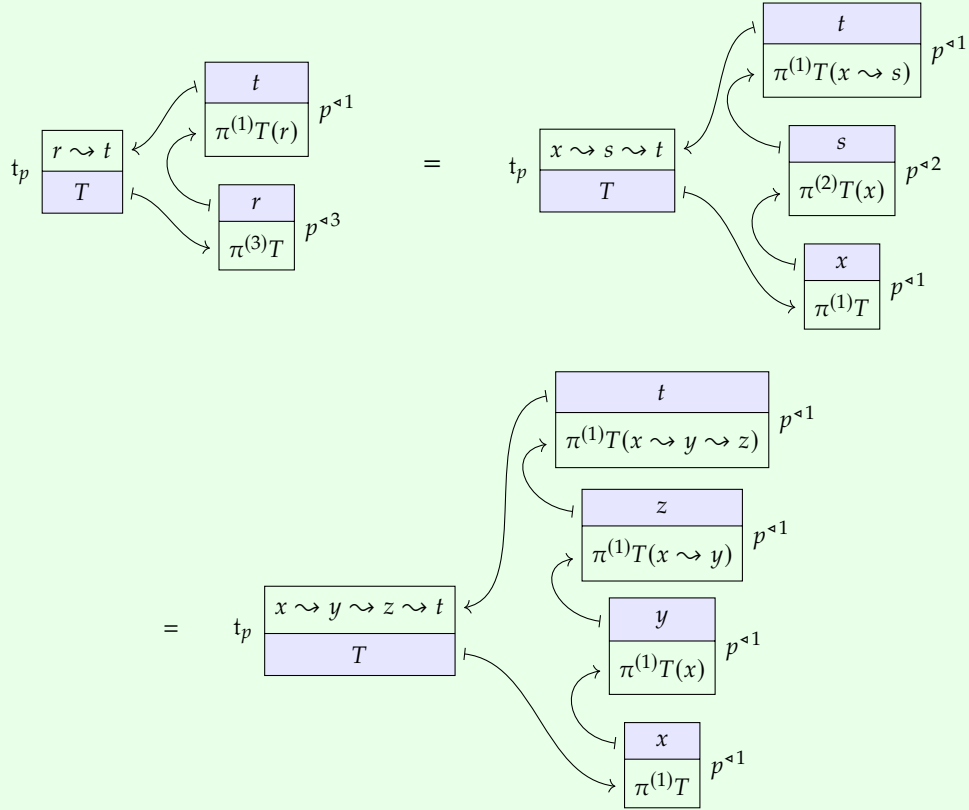


Here  $T(u)$  denotes the  $p$ -tree equal to the subtree of  $T$  rooted at its vertex  $u$ , and  $w$  is one of  $T(u)$ 's height-2 vertices. When viewed as the subtree of  $T$  rooted at its the height-2 vertex  $u$ , the  $p$ -tree  $T(u)$  has its height-2 vertex  $w$  identified with a height-4 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  $u \in \text{vtx}(T)$ , and the notation  $u \rightsquigarrow w$ , for the vertex in  $\text{vtx}(T)$  that  $w \in \text{vtx}(T(u))$  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.<sup>2</sup> As a sneak preview, they will be crucial to defining the categorical structure of the cofree comonoid on  $p$ .

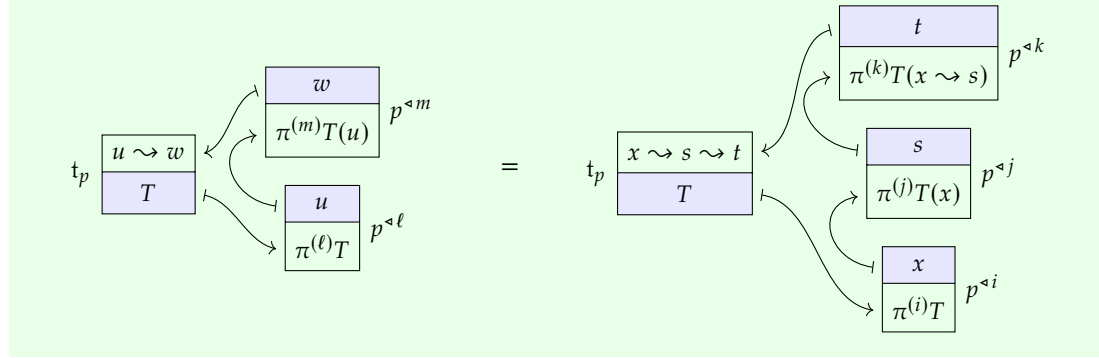
Here are three more ways to depict  $\epsilon_p^{(4)}$ , viewing its codomain in turn as  $p^{\triangleleft 3} \triangleleft p^{\triangleleft 1}$ , as  $p^{\triangleleft 1} \triangleleft p^{\triangleleft 2} \triangleleft p^{\triangleleft 1}$ , or as  $p^{\triangleleft 1} \triangleleft p^{\triangleleft 1} \triangleleft p^{\triangleleft 1} \triangleleft p^{\triangleleft 1}$ :



Here  $r = x \rightsquigarrow s$  and  $s = y \rightsquigarrow z$ . Notice that the last depiction is just another way to write (7.22), 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  $\epsilon_p^{(n)}: t_p \rightarrow p^{\triangleleft n}$ , 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  $n = i + j + k$ :



### 7.1.2 Cofree comonoids as categories

We have now characterized the carrier  $t_p$  of the comonoid  $\mathcal{T}_p$  that will turn out to be cofree on  $p$ ; but we have yet to describe the comonoid structure of  $\mathcal{T}_p$  that  $t_p$  carries. This structure will allow us to interpret  $\mathcal{T}_p$  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  $\mathcal{T}_p$ , as well as the comonoid laws that they satisfy, arise naturally from our construction of  $t_p$  as the limit of (7.1).

#### The eraser of the cofree comonoid

The eraser for a comonoid  $\mathcal{T}_p$  carried by  $t_p$  should be a lens of the form  $t_p \rightarrow y$ . Conveniently, since  $y$  appears in (7.1), its limit  $t_p$  is already equipped with a canonical lens  $\epsilon_p: t_p \rightarrow y$ , as seen in (7.19). 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  $t_p$ -position is a tree  $T \in \text{tree}_p$ , and each  $t_p[T]$ -direction is a vertex of  $T$ , the eraser  $\epsilon_p$  should pick out a single vertex of every  $p$ -tree. Even without looking at (7.19), 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 7.18 tells us that  $\epsilon_p: t_p \rightarrow y$  sends each  $p$ -tree  $T \in \text{tree}_p$  to its stage-0 pretree on positions, stripping away everything except for the single unlabeled root of  $T$ ; then on-directions,  $\epsilon_p$  picks out the unique height-0 vertex of  $T$  in  $\text{vtx}_0(T) \subseteq \text{vtx}(T)$ , which is just its root.

So in the category  $\mathcal{T}_p$ , 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  $\text{id}_T$  (so that  $\text{vtx}_0(T) = \{\text{id}_T\}$ ). Equivalently, we can identify  $\text{id}_T$  with the

<sup>2</sup>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  $\mathcal{T}_p$  will be determined by its duplicator.

### The duplicator of the cofree comonoid

The duplicator for a comonoid  $\mathcal{T}_p$  carried by  $t_p$  should be a lens  $\delta_p: t_p \rightarrow t_p \triangleleft t_p$ . But before we can specify such a lens, we need to figure out what kind of polynomial  $t_p \triangleleft t_p$  is.

Here is where our work from Sections 5.3.2 and 5.3.3 comes in handy: since the diagram in (7.1) is connected, its limit  $t_p$  is a connected limit, and we showed in Theorem 5.86 that  $\triangleleft$  preserves connected limits. Therefore  $t_p \triangleleft t_p$  is itself the limit of some diagram, and a morphism to  $t_p \triangleleft t_p$  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  $t_p \triangleleft t_p$  as the limit of the following diagram, where we have applied the functor  $- \triangleleft t_p$  to the diagram from (7.1):

$$\begin{array}{ccccccc} y \triangleleft t_p & & p \triangleleft t_p & & p^{\triangleleft 2} \triangleleft t_p & & p^{\triangleleft 3} \triangleleft t_p & & \dots \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\ 1 \triangleleft t_p & \longleftarrow & p \triangleleft 1 \triangleleft t_p & \longleftarrow & p^{\triangleleft 2} \triangleleft 1 \triangleleft t_p & \longleftarrow & p^{\triangleleft 3} \triangleleft 1 \triangleleft t_p & \longleftarrow & \dots \end{array}$$

But we have  $p^{\triangleleft n} \triangleleft 1 \triangleleft t_p \cong p^{\triangleleft n} \triangleleft 1$ , so we can simplify this diagram like so:

$$\begin{array}{ccccccc} y \triangleleft t_p & & p \triangleleft t_p & & p^{\triangleleft 2} \triangleleft t_p & & p^{\triangleleft 3} \triangleleft t_p & & \dots \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\ 1 & \longleftarrow & p \triangleleft 1 & \longleftarrow & p^{\triangleleft 2} \triangleleft 1 & \longleftarrow & p^{\triangleleft 3} \triangleleft 1 & \longleftarrow & \dots \end{array} \quad (7.26)$$

(This works because according to Proposition 5.74,  $\triangleleft$  actually preserves *all* limits on the left—including the terminal object 1.) So  $t_p \triangleleft t_p$  is the limit of (7.26).

Then we use the fact that  $\triangleleft$  preserves connected limits on the right to expand each  $p^{\triangleleft \ell} \triangleleft t_p$  in (7.26) as the limit of the following:

$$\begin{array}{ccccccc} p^{\triangleleft \ell} \triangleleft y & & p^{\triangleleft \ell} \triangleleft p & & p^{\triangleleft \ell} \triangleleft p^{\triangleleft 2} & & p^{\triangleleft \ell} \triangleleft p^{\triangleleft 3} & & \dots \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\ p^{\triangleleft \ell} \triangleleft 1 & \longleftarrow & p^{\triangleleft \ell} \triangleleft p \triangleleft 1 & \longleftarrow & p^{\triangleleft \ell} \triangleleft p^{\triangleleft 2} \triangleleft 1 & \longleftarrow & p^{\triangleleft \ell} \triangleleft p^{\triangleleft 3} \triangleleft 1 & \longleftarrow & \dots \end{array} \quad (7.27)$$

So the limit of (7.26) is the limit of a larger diagram, where we have “plugged in” a copy of (7.27) in place of each  $p^{\triangleleft \ell} \triangleleft t_p$  that appears.

It’s worth being a little careful, though, when we draw this diagram: each  $p^{\triangleleft \ell} \triangleleft t_p$  in (7.26) 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  $t_p \rightarrow 1$ . But 1 actually shows up in the lower left corner of (7.1), so the unique lens  $t_p \rightarrow 1$  is just the projection from the limit of (7.1) to that 1. Then once we apply the



connected limit-preserving functor  $p^{\triangleleft \ell} \triangleleft -$  to (7.1), yielding (7.27), we obtain the lens  $p^{\triangleleft \ell} \triangleleft t_p \rightarrow p^{\triangleleft \ell} \triangleleft 1$  we desire as the projection from the limit of (7.27) to the  $p^{\triangleleft \ell} \triangleleft 1$  in the lower left corner. So if we want to replace each  $p^{\triangleleft \ell} \triangleleft t_p$  in (7.26) with a copy of (7.27), without changing the limit of the whole diagram, we should replace the arrow  $p^{\triangleleft \ell} \triangleleft t_p \rightarrow p^{\triangleleft \ell} \triangleleft 1$  with an identity arrow from the  $p^{\triangleleft \ell} \triangleleft 1$  in the lower left corner of (7.27) to the  $p^{\triangleleft \ell} \triangleleft 1$  in (7.26). 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:

$$\begin{array}{ccccccc}
 p^{\triangleleft 3} & & p^{\triangleleft 3} \triangleleft p & & p^{\triangleleft 3} \triangleleft p^{\triangleleft 2} & & \dots \\
 \searrow & & \searrow & & \searrow & & \\
 p^{\triangleleft 3} \triangleleft 1 & & p^{\triangleleft 3} \triangleleft p \triangleleft 1 & & p^{\triangleleft 3} \triangleleft p^{\triangleleft 2} \triangleleft 1 & & \dots \\
 \downarrow & & \downarrow & & \downarrow & & \\
 p^{\triangleleft 2} & & p^{\triangleleft 2} \triangleleft p & & p^{\triangleleft 2} \triangleleft p^{\triangleleft 2} & & \dots \\
 \searrow & & \searrow & & \searrow & & \\
 p^{\triangleleft 2} \triangleleft 1 & & p^{\triangleleft 2} \triangleleft p \triangleleft 1 & & p^{\triangleleft 2} \triangleleft p^{\triangleleft 2} \triangleleft 1 & & \dots \\
 \downarrow & & \downarrow & & \downarrow & & \\
 p & & p \triangleleft p & & p \triangleleft p^{\triangleleft 2} & & \dots \\
 \searrow & & \searrow & & \searrow & & \\
 p \triangleleft 1 & & p \triangleleft p \triangleleft 1 & & p \triangleleft p^{\triangleleft 2} \triangleleft 1 & & \dots \\
 \downarrow & & \downarrow & & \downarrow & & \\
 y & & p & & p^{\triangleleft 2} & & \dots \\
 \searrow & & \searrow & & \searrow & & \\
 1 & \longleftarrow & p \triangleleft 1 & \longleftarrow & p^{\triangleleft 2} \triangleleft 1 & \longleftarrow & \dots
 \end{array} \tag{7.28}$$

Here the bottom row of (7.26) is still the bottom row of (7.28), but in the place of each  $p^{\triangleleft \ell} \triangleleft t_p$  in the top row, we have grafted in a copy of (7.27). Yet the limit of the diagram is preserved: the limit of (7.28) is still  $t_p \triangleleft t_p$ . In summary, for purely formal reasons, Proposition 5.74 and Theorem 5.86 ensure that since the limit of (7.1) is  $t_p$ , the limit of (7.28) is  $t_p \triangleleft t_p$ .

While (7.28) has become rather unwieldy to draw, it is easy to characterize: it is a diagram with a copy each of

$$p^{\triangleleft \ell} \triangleleft p^{\triangleleft m} \quad \text{and} \quad p^{\triangleleft \ell} \triangleleft p^{\triangleleft m} \triangleleft 1$$

for every  $\ell, m \in \mathbb{N}$ , with arrows

$$p^{\triangleleft \ell} \triangleleft p^{\triangleleft m} \rightarrow p^{\triangleleft \ell} \triangleleft p^{\triangleleft m} \triangleleft 1 \tag{7.29}$$

between them; along with arrows

$$p^{\triangleleft \ell} \triangleleft p^{\triangleleft (m+1)} \triangleleft 1 \rightarrow p^{\triangleleft \ell} \triangleleft p^{\triangleleft m} \triangleleft 1, \tag{7.30}$$

drawn vertically in (7.28); and

$$p^{\triangleleft \ell+1} \triangleleft p^{\triangleleft 0} \triangleleft 1 \cong p^{\triangleleft \ell+1} \triangleleft 1 \rightarrow p^{\triangleleft \ell} \triangleleft 1 \cong p^{\triangleleft \ell} \triangleleft p^{\triangleleft 0} \triangleleft 1, \tag{7.31}$$

drawn horizontally in (7.28) 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  $\ell$  rows above the bottom and  $m$  columns to the right in (7.28), 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  $t_p \rightarrow t_p \triangleleft t_p$ , by giving a map from the diagram (7.1) to the diagram (7.28) 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 (7.1) to the  $p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}$  that appears in (7.28), and similarly sending  $p^{\triangleleft n} \triangleleft 1$  in (7.1) to  $p^{\triangleleft \ell} \triangleleft p^{\triangleleft m} \triangleleft 1$  in (7.28). Then all the arrows that appear in (7.28)—i.e. all the arrows in (7.29), (7.30), and (7.31)—can be identified with arrows that appear in (7.1), so everything commutes. We therefore induce a lens from the limit  $t_p$  of (7.1) to the limit  $t_p \triangleleft t_p$  of (7.28), which we call  $\delta_p: t_p \rightarrow t_p \triangleleft t_p$ . This turns out to be the duplicator of  $\mathcal{T}_p$ .

How does  $\delta_p$  behave in terms of  $p$ -trees? First, we characterize  $t_p \triangleleft t_p$  and its projections to each  $p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}$  in (7.28). Concretely, we know that a  $t_p \triangleleft t_p$ -position is just a  $p$ -tree  $T \in \text{tree}_p$  along with a  $p$ -tree  $U(v) \in \text{tree}_p$  associated with every vertex  $v \in \text{vtx}(T)$ . Then a direction at that position is a choice of vertex  $v \in \text{vtx}(T)$  and another vertex  $w \in \text{vtx}(U(v))$  of the  $p$ -tree  $U(v)$  associated with  $v$ . Each projection from  $t_p \triangleleft t_p$  to  $p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}$  in (7.28) is then the composition product of the projections  $\epsilon_p^{(\ell)}: t_p \rightarrow p^{\triangleleft \ell}$  and  $\epsilon_p^{(m)}: t_p \rightarrow p^{\triangleleft m}$ , which by Example 7.20 we can draw in polyboxes like so:

$$\begin{array}{ccc}
 & t_p & p^{\triangleleft m} \\
 \text{vtx}(-) & \boxed{w} & \xleftarrow{\quad} \boxed{w} \text{vtx}_m(-) \\
 \text{tree}_p & \boxed{U_v} & \xrightarrow{\quad} \boxed{\pi^{(m)}U_v} p^{\triangleleft m}(1)
 \end{array}$$
  

$$\begin{array}{ccc}
 \text{vtx}(-) & \boxed{v} & \xleftarrow{\quad} \boxed{v} \text{vtx}_\ell(-) \\
 \text{tree}_p & \boxed{T} & \xrightarrow{\quad} \boxed{\pi^{(\ell)}T} p^{\triangleleft \ell}(1) \\
 & t_p & p^{\triangleleft \ell}
 \end{array}$$

On positions,  $\epsilon_p^{(\ell)} \triangleleft \epsilon_p^{(m)}$  sends the  $p$ -tree  $T$  to its stage- $\ell$  pretree  $\pi^{(\ell)}T$ , and it sends each  $p$ -tree  $U_v$  associated to a height- $\ell$  vertex  $v \in \text{vtx}_\ell(T)$  to its stage- $m$  pretree  $\pi^{(m)}U_v$ , to be  $p^{\triangleleft m}$ -pretree associated with the height- $\ell$  leaf  $v$  of  $\pi^{(m)}T$ . Equivalently, this specifies a  $p^{\triangleleft (\ell+m)}$ -pretree on the right: its bottom  $\ell$  levels coincide with the bottom  $\ell$  levels of  $T$ , and its top  $m$  levels coincide with the bottom  $m$  levels of the  $U_v$ 's for  $v \in \text{vtx}_\ell(T)$ . Then on directions,  $\epsilon_p^{(\ell)} \triangleleft \epsilon_p^{(m)}$  is the canonical inclusion of vertices  $\text{vtx}_\ell(T) \hookrightarrow \text{vtx}(T)$  sending  $v \mapsto v$ , followed by the canonical inclusion of vertices  $\text{vtx}_m(U_v) \hookrightarrow \text{vtx}(U_v)$ . These lenses comprise the universal cone over (7.28).

Meanwhile, the other cone we formed over (7.28) is comprised of lenses of the form  $\epsilon_p^{(\ell+m)}: t_p \rightarrow p^{\triangleleft (\ell+m)} \cong p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}$ , each of which should factor through  $t_p \triangleleft t_p$ , as depicted

in the following commutative diagram:

$$\begin{array}{ccc}
 t_p & \xrightarrow{\delta_p} & t_p \triangleleft t_p \\
 \searrow \epsilon_p^{(\ell+m)} & & \downarrow \epsilon_p^{(\ell)} \triangleleft \epsilon_p^{(m)} \\
 & & p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}.
 \end{array} \tag{7.32}$$

Indeed, by the universal property of  $t_p \triangleleft t_p$ , our  $\delta_p$  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 (7.32) repeatedly in what follows, whenever we work with  $\delta_p$ .

Expressing (7.32) as an equation of polyboxes, using our usual labels for the arrows of the duplicator  $\delta_p$  on the left and our depiction of the projection  $\epsilon_p^{(\ell+m)}: t_p \rightarrow p^{\triangleleft \ell} \triangleleft p^{\triangleleft m}$  from Example 7.20 on the right, we have

Recall from Example 7.20 that  $T(v)$  denotes the  $p$ -tree equal to the subtree of  $T$  rooted at  $v \in \text{vtx}(T)$ , while  $v \rightsquigarrow w \in \text{vtx}(T)$  for  $w \in \text{vtx}(T(v))$  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  $v \in \text{vtx}(T)$  with the rooted path in  $T$  that ends at the vertex  $v$  and  $w \in \text{vtx}(T(v))$  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  $\text{cod } v := T(v)$  and  $v ; w := v \rightsquigarrow w$ ; in fact,  $\text{cod } v$  and  $v ; w$  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  $\mathcal{T}_p$  yields the following result.

**Proposition 7.33.** As defined above,  $(t_p, \epsilon_p, \delta_p)$  is a polynomial comonoid corresponding to a category  $\mathcal{T}_p$  characterized as follows.

- An object in  $\mathcal{T}_p$  is a  $p$ -tree in  $T \in \text{tree}_p$ .
- 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  $\text{cod } v$ , 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  $v \circ w$ , 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  $\mathcal{T}_p$ , or equivalently that the comonoid laws hold for  $(t_p, \epsilon_p, \delta_p)$ . 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

$$\begin{array}{ccc} y \triangleleft t_p & \xlongequal{\quad} & t_p \\ & \swarrow \epsilon_p \triangleleft t_p & \downarrow \delta_p \\ & & t_p \triangleleft t_p \end{array}$$

commutes. We know the lens  $\epsilon_p \triangleleft t_p : t_p \triangleleft t_p \rightarrow y \triangleleft t_p$  is characterized by its components  $\epsilon_p \triangleleft \epsilon_p^{(n)} : t_p \triangleleft t_p \rightarrow p^{\triangleleft 0} \triangleleft p^{\triangleleft n}$ , a projection out of the limit  $t_p \triangleleft t_p$ , for all  $n \in \mathbb{N}$ . Then by our construction of  $\delta_p$ , each composite  $\delta_p \circ (\epsilon_p \triangleleft \epsilon_p^{(n)}) : t_p \rightarrow p^{\triangleleft 0} \triangleleft p^{\triangleleft n}$  is the component of  $\delta_p$  equal to  $\epsilon_p^{(n)} = \epsilon_p^{(0+n)} : t_p \rightarrow p^{\triangleleft 0} \triangleleft p^{\triangleleft n} \cong p^{\triangleleft n}$  (this is just the commutativity of (7.32) in the case of  $(\ell, m) = (0, n)$ ). Together, these characterize the composite lens  $\delta_p \circ (\epsilon_p \triangleleft t_p) : t_p \rightarrow y \triangleleft t_p \cong t_p$  as the lens whose components are the projections  $\epsilon_p^{(n)} : t_p \rightarrow p^{\triangleleft n}$  from the limit. It follows from the universal property of  $t_p$  that  $\delta_p \circ (\epsilon_p \triangleleft t_p)$  can only be the identity lens on  $t_p$ . Hence the left erasure law holds.

The right erasure law, that

$$\begin{array}{ccc} t_p & \xlongequal{\quad} & t_p \triangleleft y \\ \delta_p \downarrow & \nearrow t_p \triangleleft \epsilon_p & \\ t_p \triangleleft t_p & & \end{array}$$

commutes, follows similarly: the composite  $\delta_p \circ (t_p \triangleleft \epsilon_p)$  is characterized by its components over  $n \in \mathbb{N}$  of the form  $\delta_p \circ (\epsilon_p^{(n)} \triangleleft \epsilon_p) : t_p \rightarrow p^{\triangleleft n} \triangleleft p^{\triangleleft 0}$ , which we know by (7.32) 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  $\delta_p \circ (t_p \triangleleft \epsilon_p)$  must be the identity lens on  $t_p$ .

Finally, we check the coassociative law: that

$$\begin{array}{ccc} t_p & \xrightarrow{\delta_p} & t_p \triangleleft t_p \\ \delta_p \downarrow & & \downarrow t_p \triangleleft \delta_p \\ t_p \triangleleft t_p & \xrightarrow{\delta_p \triangleleft t_p} & t_p \triangleleft t_p \triangleleft t_p, \end{array}$$

commutes. Because  $\triangleleft$  preserves connected limits, we can write  $t_p \triangleleft t_p \triangleleft t_p$  as a limit the way we did with  $t_p \triangleleft t_p$ : it is the limit of diagram consisting of arrows

$$p^{\triangleleft i} \triangleleft p^{\triangleleft j} \triangleleft p^{\triangleleft k} \rightarrow p^{\triangleleft i} \triangleleft p^{\triangleleft j} \triangleleft p^{\triangleleft k} \triangleleft 1$$

for each  $i, j, k \in \mathbb{N}$ , with additional arrows between the position-sets. So a lens to  $t_p \triangleleft t_p \triangleleft t_p$ , 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 (7.32),

$$\begin{aligned} \delta_p \circ (t_p \triangleleft \delta_p) \circ (\epsilon_p^{(i)} \triangleleft \epsilon_p^{(j)} \triangleleft \epsilon_p^{(k)}) &= \delta_p \circ (\epsilon_p^{(i)} \triangleleft (\delta_p \circ (\epsilon_p^{(j)} \triangleleft \epsilon_p^{(k)}))) \\ &= \delta_p \circ (\epsilon_p^{(i)} \triangleleft \epsilon_p^{(j+k)}) \end{aligned} \quad (7.32)$$

$$= \epsilon_p^{(i+j+k)} \quad (7.32)$$

$$= \delta_p \circ (\epsilon_p^{(i+j)} \triangleleft \epsilon_p^{(k)}) \quad (7.32)$$

$$= \delta_p \circ ((\delta_p \circ (\epsilon_p^{(i)} \triangleleft \epsilon_p^{(j)})) \triangleleft \epsilon_p^{(k)}) \quad (7.32)$$

$$= \delta_p \circ (\delta_p \triangleleft t_p) \circ (\epsilon_p^{(i)} \triangleleft \epsilon_p^{(j)} \triangleleft \epsilon_p^{(k)}).$$

for all  $i, j, k \in \mathbb{N}$ . So by the universal property of  $t_p \triangleleft t_p \triangleleft t_p$ , coassociativity holds.  $\square$

We call the category  $\mathcal{T}_p$  corresponding to the comonoid  $(t_p, \epsilon_p, \delta_p)$  the *category of  $p$ -trees*, the *category of trees on  $p$* , or the  *$p$ -tree category*.

*Example 7.34* (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 an arena of positions and directions. 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  $p^{\triangleleft 2}(1) \rightarrow p^{\triangleleft 1}(1)$ .

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 (7.1) 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  $t_p$  of (7.1) 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  $t_p$ , completing the cofree construction of the category of  $p$ -trees  $\mathcal{T}_p$ .

Soon we will prove that  $\mathcal{T}_p$  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 7.33 whenever convenient.

*Example 7.35* (The category of 1-trees). Let's start by taking  $p := 1$ . In Exercise 7.17 #1, we showed that there is a unique 1-tree: a tree with only 1 vertex, its root. So  $t_1 \cong y$  is the carrier of the category of 1-trees  $\mathcal{T}_1$ .

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  $\bullet \rightarrow \_$  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  $\text{id}_\bullet: \bullet \rightarrow \bullet$  is the root of  $\bullet$ . This is the same morphism we just mentioned. Equivalently, it is the empty rooted path.
- The composite morphism  $\text{id}_\bullet \circ \text{id}_\bullet = \text{id}_\bullet \circ \text{id}_{\text{cod}(\text{id}_\bullet)}: \bullet \rightarrow \_$  is obtained by concatenating the empty rooted path  $\text{id}_\bullet$  of  $\bullet$  with the empty rooted path  $\text{id}_{\text{cod}(\text{id}_\bullet)}$  of  $\text{cod}(\text{id}_\bullet)$ , which is again just the empty rooted path of  $\bullet$ . Hence  $\text{id}_\bullet \circ \text{id}_{\text{cod} \text{id}_\bullet} = \text{id}_\bullet$ , 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 7.36* (The category of trees on a constant). Let  $B$  be a set, viewed as a constant polynomial.

1. What is the polynomial  $t_B$ ?
2. Characterize the  $B$ -tree category  $\mathcal{T}_B$ . ◇

*Example 7.37* (The category of  $y$ -trees). Now consider  $p := y$ . In Exercise 7.17 #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  $t_y \cong y^{\mathbb{N}}$  is the carrier of the category of  $y$ -trees  $\mathcal{T}_y$ . 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 6.40 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  $\mathcal{T}_y$ ?

We can characterize  $\mathcal{T}_y$  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  $\text{id}_{\uparrow}: \uparrow \rightarrow \uparrow$  is the empty rooted path, which has length 0; so  $\text{id}_{\uparrow} = 0$ . In the corresponding monoid structure on  $\mathbb{N}$ , the element  $0 \in \mathbb{N}$  must then be the unit.
- The composite morphism  $m \circ n: \uparrow \rightarrow \uparrow$  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  $m \circ n = m + n$ . In the corresponding monoid structure on  $\mathbb{N}$ , the binary operation must then be given by addition.

Hence  $\mathcal{T}_y$  is the monoid  $(\mathbb{N}, 0, +)$  viewed as a 1-object category.

*Example 7.38* ( $By$ -trees are  $B$ -streams). Let  $B$  be a set, and consider  $p := By$ . Generalizing Exercise 7.17 #5 for  $B$  instead of 2, or applying Exercise 7.17 #7 for the case of  $A := 1$ , 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  $\mathbb{N} \rightarrow B$  assigning each vertex a label. Hence  $t_{By} \cong B^{\mathbb{N}}y^{\mathbb{N}}$  is the carrier of the category of  $By$ -trees  $\mathcal{T}_{By}$ . As in Example 7.37, 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  $\mathcal{T}_{By}$  once before: it is the category of  $B$ -streams from Example 6.45.

- 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:

$$\bar{b} := (b_0 \rightarrow b_1 \rightarrow b_2 \rightarrow b_3 \rightarrow \cdots).$$

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.

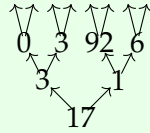
- Given a  $B$ -stream  $\bar{b} \in B^{\mathbb{N}}$  above, recall that a morphism out of  $\bar{b}$  is a natural number  $n \in \mathbb{N}$ , and its codomain is the substream of  $B$  starting at  $b_n$ :

$$\text{cod}(\bar{b} \xrightarrow{n} \_) = (b_n \rightarrow b_{n+1} \rightarrow b_{n+2} \rightarrow b_{n+3} \rightarrow \cdots).$$

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  $\bar{b}$  as a  $B$ -stream or a  $By$ -tree, its identity morphism  $\text{id}_{\bar{b}}: \bar{b} \rightarrow \bar{b}$  corresponds to  $0 \in \mathbb{N}$ ; from the latter perspective, it is the length-0 path from the root to itself.
- Whether we view  $\bar{b}$  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 7.39* (The category of  $\mathbb{N}$ -labeled binary trees). Consider  $p := \mathbb{N}y^2$ . By Exercise 7.17 #7, or Exercise 7.12 #2 in the case of  $L := \mathbb{N}$  and  $n := 2$ , 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  $\mathcal{T}_{\mathbb{N}y^2}$  as follows.

- The objects are  $\mathbb{N}$ -labeled binary trees from the set  $\mathbb{N}^{\text{List}(2)}$ .
- A morphism out of an  $\mathbb{N}$ -labeled binary tree is a binary sequence: a finite list of directions in 2, whose elements could be interpreted as “left” and “right,” thus uniquely specifying a rooted path in a binary tree. They comprise the set  $\text{List}(2)$ . 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 7.40* (The category of  $B$ -labeled  $A$ -ary trees). Characterize the  $By^A$ -tree category  $\mathcal{T}_{By^A}$ .  $\diamond$

*Exercise 7.41.* Characterize the  $(y + 1)$ -tree category  $\mathcal{T}_{y+1}$ . ◇

*Exercise 7.42.* Let  $p := \{a, b, c, \dots, z, \sqcup\}y + \{\bullet\}$ .

1. Describe the objects of the cofree category  $\mathcal{T}_p$ , 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 7.43.* Let  $p := \{\bullet, \bullet\}y^2 + \{\bullet\}y + \{\bullet\}$  as in Example 7.10.

1. Choose an object  $t \in \text{tree}_p$ , 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 7.44.* Let  $p$  be a polynomial, let  $\mathbb{Q} := \{q \in \mathbb{Q} \mid q \geq 0\}$  and consider the monoid  $y^{\mathbb{Q}}$  of nonnegative rational numbers under addition. Is it true that any cofunctor  $\varphi: \mathcal{T}_p \rightarrow y^{\mathbb{Q}}$  is constant, i.e. that it factors as

$$\mathcal{T}_p \rightarrow y \rightarrow y^{\mathbb{Q}}?$$
◇

### 7.1.3 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  $\mathcal{T}_p$  we constructed is the cofree comonoid on  $p$ .

**Theorem 7.45** (Cofree comonoid). The forgetful functor  $U: \mathbf{Cat}^\# \rightarrow \mathbf{Poly}$  has a right adjoint  $\mathcal{T}_-: \mathbf{Poly} \rightarrow \mathbf{Cat}^\#$ , giving rise to an adjunction

$$\mathbf{Cat}^\# \begin{array}{c} \xrightarrow{U} \\ \xRightarrow{\quad} \\ \xleftarrow{\mathcal{T}_-} \end{array} \mathbf{Poly} ,$$

such that for each  $p \in \mathbf{Poly}$ , the carrier  $t_p := U\mathcal{T}_p$  of the category  $\mathcal{T}_p$  is given by the limit of the diagram (7.1), repeated here:

$$\begin{array}{ccccccc} y & & p & & p^{\triangleleft 2} & & p^{\triangleleft 3} & & \dots \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\ 1 & \xleftarrow{!} & p \triangleleft 1 & \xleftarrow{p \triangleleft !} & p^{\triangleleft 2} \triangleleft 1 & \xleftarrow{p^{\triangleleft 2} \triangleleft !} & p^{\triangleleft 3} \triangleleft 1 & \xleftarrow{\quad} & \dots \end{array}$$

That is, for any category  $\mathcal{C} \in \mathbf{Cat}^\#$  with carrier  $c := U\mathcal{C}$ , there is a natural isomorphism

$$\mathbf{Poly}(c, p) \cong \mathbf{Cat}^\#(\mathcal{C}, \mathcal{T}_p).$$

*Proof.* To show that  $\mathcal{T}_-$  is the right adjoint of  $U$ , it is enough to show that for every lens  $\varphi: c \rightarrow p$ , there exists a unique cofunctor  $F: \mathcal{C} \rightarrow \mathcal{T}_p$  for which

$$\begin{array}{ccc} c & & \\ \downarrow F & \searrow \varphi & \\ t_p & \xrightarrow{\epsilon_p^{(1)}} & p \end{array} \quad (7.46)$$

commutes. Here the projection  $\epsilon_p^{(1)}: U\mathcal{T}_p \cong t_p \rightarrow p$  serves as the counit of the adjunction, and we identify the cofunctor  $F$  with its underlying lens  $UF: c \rightarrow t_p$ .

First, we construct  $F$  from  $\varphi$  as follows. If we let  $\epsilon$  and  $\delta$  be the eraser and duplicator of  $\mathcal{C}$ , the diagram

$$\begin{array}{ccc} c & \xrightarrow{\delta} & c \triangleleft c \\ \epsilon \downarrow & \searrow \varphi & \downarrow \epsilon \triangleleft \varphi \\ y & & p \\ \downarrow & & \downarrow \\ 1 & \xleftarrow{\quad} & p \triangleleft 1 \end{array} \quad (7.47)$$

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

$$\begin{aligned} \delta \circ (\epsilon \triangleleft \varphi) &= \delta \circ (\epsilon \triangleleft c) \circ \varphi \\ &= \text{id}_c \circ \varphi \\ &= \varphi. \end{aligned} \quad (\text{Left erasure law})$$

Then by induction, the larger diagram

$$\begin{array}{ccccccc}
 c & \xrightarrow{\delta} & c \triangleleft c & \xrightarrow{\delta \triangleleft c} & c^{\triangleleft 3} & \xrightarrow{\delta \triangleleft c^{\triangleleft 2}} & c^{\triangleleft 4} \longrightarrow \dots \\
 \epsilon \downarrow & \searrow \varphi & \downarrow \epsilon \triangleleft \varphi & & \downarrow \epsilon \triangleleft \varphi^{\triangleleft 2} & & \downarrow \epsilon \triangleleft \varphi^{\triangleleft 3} \\
 y & & p & & p^{\triangleleft 2} & & p^{\triangleleft 3} \longrightarrow \dots \\
 \downarrow & & \downarrow & & \downarrow & & \downarrow \\
 1 & \longleftarrow & p \triangleleft 1 & \longleftarrow & p^{\triangleleft 2} \triangleleft 1 & \longleftarrow & p^{\triangleleft 3} \triangleleft 1 \longleftarrow \dots
 \end{array} \quad (7.48)$$

commutes as well: its leftmost rectangle is (7.47), and taking the composition product of each rectangle in (7.48) with the commutative rectangle

$$\begin{array}{ccc}
 c & \xlongequal{\quad} & c \\
 \varphi \downarrow & & \downarrow \varphi \\
 p & & p \\
 \parallel & & \parallel \\
 p & \xlongequal{\quad} & p
 \end{array}$$

yields the rectangle to its right. As  $t_p$  is the limit of the bottom two rows of (7.48), it follows that there is an induced lens  $F: c \rightarrow t_p$  that, when composed with each projection  $\epsilon_p^{(n)}: t_p \rightarrow p^{\triangleleft n}$ , yields the lens depicted in (7.48) from  $c$  to  $p^{\triangleleft n}$ . This lens is the composite of the lens  $c \rightarrow c^{\triangleleft(n+1)}$  in the top row, which by Proposition 6.20 is the canonical lens  $\delta^{(n+1)}$  associated with the comonoid  $C$ , composed with the lens  $\epsilon \triangleleft \varphi^{\triangleleft n}: c^{\triangleleft(n+1)} \rightarrow p^{\triangleleft n}$ .

Next, we prove uniqueness: that a cofunctor  $C \rightarrow \mathcal{T}_p$  with underlying lens  $f: c \rightarrow t_p$  is completely determined by the value of  $f \circ \epsilon_p^{(1)}$ . It suffices to show that we can recover the  $n^{\text{th}}$  component of  $g$  from its first component. \*\*

□

### 7.1.4 The many (inter)faces of the cofree comonoid

The forgetful-cofree adjunction of Theorem 7.45 tells us that given a category  $C \in \mathbf{Cat}^\sharp$  with carrier  $c := UC$  and a polynomial  $p \in \mathbf{Poly}$ , there is a natural isomorphism

$$\mathbf{Poly}(c, p) \cong \mathbf{Cat}^\sharp(C, \mathcal{T}_p).$$

So every lens  $\varphi: c \rightarrow p$  has a corresponding cofunctor  $F: C \rightarrow \mathcal{T}_p$  that we call its *mate*.

We can view  $\varphi$  as a dynamical system with an interface  $p$  and a generalized state system  $c$ , carrying an arbitrary category  $C$  of states and transitions. Then the lens  $\text{Run}_n(\varphi): c \rightarrow p^{\triangleleft n}$  for  $n \in \mathbb{N}$ , defined as the composite

$$c \xrightarrow{\delta^{(n)}} c^{\triangleleft n} \xrightarrow{\varphi^{\triangleleft n}} p^{\triangleleft n},$$

models  $n$  runs through the system  $\varphi$ .

Meanwhile, as the limit of the diagram (7.1) with the row of polynomials of the form  $p^{\triangleleft n}$  across the top, the carrier  $t_p$  of the cofree comonoid on  $p$  comes equipped with a lens  $\epsilon_p^{(n)}: t_p \rightarrow p^{\triangleleft n}$  for each  $n \in \mathbb{N}$ . Since the cofunctor  $F: \mathcal{C} \rightarrow \mathcal{T}_p$  has an underlying lens between carriers  $f := UF: c \rightarrow t_p$ , we can obtain another lens  $c \rightarrow^{\triangleleft n} p^{\triangleleft n}$  as the composite

$$c \xrightarrow{f} t_p \xrightarrow{\epsilon_p^{(n)}} p^{\triangleleft n}.$$

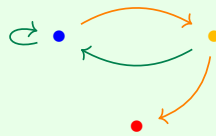
It follows from our forgetful-cofree adjunction that these two composites are equal.

**Proposition 7.49.** With the definitions above, the following diagram commutes:

$$\begin{array}{ccc} c & \xrightarrow{f} & t_p \\ \delta^{(n)} \downarrow & & \downarrow \epsilon_p^{(n)} \\ c^{\triangleleft n} & \xrightarrow{\varphi^{\triangleleft n}} & p^{\triangleleft n}. \end{array}$$

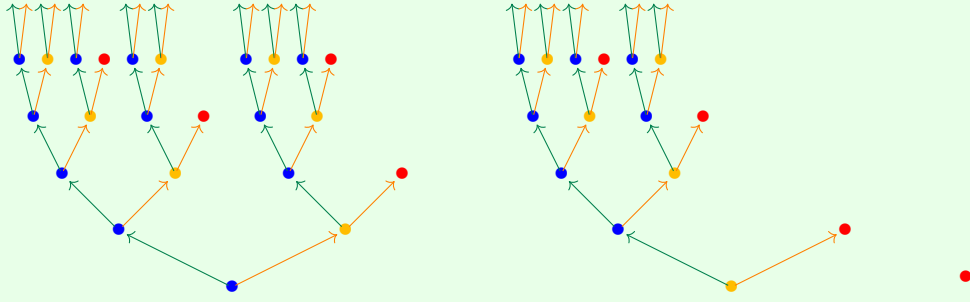
Now we have a better sense of what we mean when we say that  $F: \mathcal{C} \rightarrow \mathcal{T}_p$  captures all the information that the lenses  $\text{Run}_n(\varphi): c \rightarrow p^{\triangleleft n}$  encode: the category  $\mathcal{T}_p$  is carried by a polynomial  $t_p$  equipped with lenses  $\epsilon_p^{(n)}: t_p \rightarrow p^{\triangleleft n}$ , each of which exposes a part of the category as the  $n$ -fold interface  $p^{\triangleleft n}$ . All together,  $t_p$  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 7.50.* Let  $S := \{\bullet, \circ, \circ\}$  and  $p := y^2 + 1$ , and consider the dynamical system  $\varphi: Sy^S \rightarrow p$  modeling the halting deterministic state automaton from Exercise 3.21, depicted here again for your convenience:



Under the forgetful-cofree adjunction, the lens  $\varphi$  coincides with a cofunctor  $F: Sy^S \rightarrow \mathcal{T}_p$  from the state category on  $S$  to the category of  $p$ -trees. The cofunctor sends each state in  $S = \{\bullet, \circ, \circ\}$  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  $\varphi: Sy^S \rightarrow y^A + 1$  modeling a halting deterministic state automaton and an initial state  $s_0 \in S$ , the  $(y^A + 1)$ -tree to which the corresponding cofunctor  $F: Sy^S \rightarrow \mathcal{T}_{y^A+1}$  sends  $s_0$  encodes the set of words accepted by the automaton with that initial state in its maximal rooted paths.

*Example 7.51* (Languages recognized by deterministic state automata). Recall from Proposition 3.16 that a deterministic state automaton with a set of states  $S$  and a set of input symbols  $A$  can be identified with a lens  $y \rightarrow Sy^S$ , indicating the initial state  $s_0 \in S$ , and a lens  $\varphi: Sy^S \rightarrow 2y^A$ , indicating the subset of accept states  $F := \varphi_1^{-1}(2) \subseteq S$  and the update function  $u: S \times A \rightarrow S$  via  $u(s, a) = \varphi_s^\sharp(a)$ .

Under the forgetful-cofree adjunction, the lens  $\varphi$  coincides with a cofunctor  $F: Sy^S \rightarrow \mathcal{T}_{2y^A}$ . By Exercise 7.40,  $\mathcal{T}_{2y^A}$  is the category of 2-labeled  $A$ -ary trees. So  $F(s_0)$  is an element of  $\text{tree}_{2y^A} \cong 2^{\text{List}(A)}$ : 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  $2^{\text{List}(A)}$  is a subset of  $\text{List}(A)$ ; in the case of  $F(s_0)$ , it is the subset containing exactly the words in  $\text{List}(A)$  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 7.52* (Input sequences to output sequences). We interpret our dynamical systems as converting sequences of input to sequences of output. 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  $\varphi: Sy^S \rightarrow By^A$ , although of course we can generalize this beyond monomial interfaces.

The lens  $\varphi$  corresponds to a cofunctor  $F: Sy^S \rightarrow \mathcal{T}_{By^A}$ . By Exercise 7.40,  $\mathcal{T}_{By^A}$  is the category of  $B$ -labeled  $A$ -ary trees; in particular, its carrier is  $t_{By^A} \cong B^{\text{List}(A)} y^{\text{List}(A)}$ .

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 inputs that the system may receive. The label in  $B$  corresponding to the vertex (or rooted path) specified by each sequence in  $\text{List}(A)$  tells us the final output that the system returns when that sequence is fed in as input. Put another way,  $F(s_0) \in B^{\text{List}(A)}$  can be interpreted as a function  $\text{List}(A) \rightarrow B$ . So if the input sequence is  $(a_1, \dots, a_n) \in A^n \subseteq \text{List}(A)$ , then the corresponding output sequence  $(b_0, \dots, b_n) \in B^{n+1}$  is given (non-recursively!) by

$$b_i := F(s_0)(a_1, \dots, a_i).$$

Finally,  $F_{s_0}^\#(a_1, \dots, a_i) \in S$  then corresponds to the system's state after that sequence of inputs is given.

### 7.1.5 Morphisms between cofree comonoids

Given a morphism of polynomials  $\varphi: p \rightarrow q$ , the cofree functor gives us a map of comonoids  $\mathcal{T}_\varphi: \mathcal{T}_p \rightarrow \mathcal{T}_q$ , which works as follows.

An object  $t \in \text{tree}_p$  is a tree; the tree  $u := \mathcal{T}_\varphi(t) \in \text{tree}_q$  is constructed recursively as follows. If the root of  $t$  is  $i \in p(1)$  then the root of  $u$  is  $j := \varphi_1(i)$ . To each branch  $b \in q[j]$ , we need to assign a new tree, and we use the one situated at  $\varphi_i^\#(b)$ .

*Exercise 7.53.* Let  $p := \{\bullet\}y^2 + \{\bullet\}$  and  $q := \{\bullet, \bullet\}y + \{\bullet, \bullet\}$ .

1. Choose a map  $\varphi: p \rightarrow q$ , and write it out.
2. Choose a tree  $T \in \text{tree}_p$  with at least height 3.
3. What is  $\mathcal{T}_\varphi(T)$ ?

◇

*Exercise 7.54.* Let  $p$  be a polynomial.

1. Show there is an induced cofunctor  $\mathcal{T}_p \rightarrow \mathcal{T}_{p^{\ast n}}$  for all  $n \in \mathbb{N}$ .
2. When  $n \geq 1$ , is the induced cofunctor is an isomorphism?

◇

### 7.1.6 Some categorical properties of cofree comonoids

**Proposition 7.55.** For every polynomial  $p$ , the cofree category  $\mathcal{T}_p$  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  $\mathcal{T}_p$ .

*Proof.* For vertices, we let  $V_p$  denote the set of  $p$ -trees,

$$V_p := \text{tree}_p(1).$$

For arrows we use the counit map  $\pi: \text{tree}_p \rightarrow p$  from Theorem 7.45 to define

$$A_p := \sum_{t \in \text{tree}_p(1)} p[\pi_1(t)]$$

In other words  $A_p$  is the set  $\{d \in p[\pi_1(t)] \mid t \in \text{tree}_p\}$  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  $\text{cod}(\pi_t^\sharp(d))$ . It is clear that every morphism in  $\mathcal{T}_t$  is the composite of a finite sequence of such morphisms, completing the proof.  $\square$

**Corollary 7.56.** Let  $p$  be a polynomial and  $\mathcal{T}_p$  the cofree comonoid. Every morphism in  $\mathcal{C}_p$  is both monic and epic.

*Proof.* The free category on a graph always has this property, so the result follows from Proposition 7.55.  $\square$

**Proposition 7.57.** The additive monoid  $y^{\mathbb{N}}$  of natural numbers has a  $\times$ -monoid structure in  $\mathbf{Cat}^\sharp$ .

*Proof.* The right adjoint  $p \mapsto \mathcal{T}_p$  preserves products, so  $y^{\text{List}(n)} \cong \mathcal{T}_{y^n}$  is the  $n$ -fold product of  $y^{\mathbb{N}}$  in  $\mathbf{Cat}^\sharp$ . We thus want to find cofunctors  $e: y \rightarrow y^{\mathbb{N}}$  and  $m: y^{\text{List}(2)} \rightarrow y^{\mathbb{N}}$  that satisfy the axioms of a monoid.

The unique polynomial map  $y \rightarrow y^{\mathbb{N}}$  is a cofunctor (it is the mate of the identity  $y \rightarrow y$ ). We take  $m$  to be the mate of the polynomial map  $y^{\text{List}(2)} \rightarrow y$  given by the list  $[1, 2]$ . One can check by hand that these definitions make  $(y^{\mathbb{N}}, e, m)$  a monoid in  $(\mathbf{Cat}^\sharp, y, \times)$ .  $\square$

Recall from Example 6.70 that an arrow field of a category  $\mathcal{C}$  is a cofunctor  $\mathcal{C} \rightarrow y^{\mathbb{N}}$ .

**Corollary 7.58.** 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

$$\mathbf{Cat}^\sharp(-, y^{\mathbb{N}}): \mathbf{Cat}^\sharp \rightarrow \mathbf{Mon}^{\text{op}}$$

*Proof.* We saw that  $y^{\mathbb{N}}$  is a monoid object in Proposition 7.57.  $\square$

A cofunctor  $\mathcal{C} \rightarrow y^{\mathbb{N}}$  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 7.59.** The arrow fields functor

$$\mathbf{Cat}^\sharp \rightarrow \mathbf{Mon}^{\text{op}}$$

is right adjoint to the inclusion  $\mathbf{Mon}^{\text{op}} \rightarrow \mathbf{Cat}^\sharp$  from Proposition 6.78.

*Proof.* Let  $\mathcal{C}$  be a category and  $(M, e, *)$  a monoid. A cofunctor  $F: \mathcal{C} \rightarrow y^M$  has no data on objects; it is just a way to assign to each  $c \in \mathcal{C}$  and  $m \in M$  a morphism  $F_c^\sharp(m): c \rightarrow c'$  for some  $c' := \text{cod}(F_c^\sharp(m))$ . This assignment must send identities to identities and composites to composites: given  $m' \in M$  we have  $F_c^\sharp(m \circ m') = F_c^\sharp(m) \circ F_c^\sharp(m')$ . This is exactly the data of a monoid morphism  $M \rightarrow \vec{\mathcal{C}}$ : it assigns to each  $m \in M$  an arrow field  $\mathcal{C}$ , preserving unit and multiplication.  $\square$

**Proposition 7.60.** There is a commutative square of left adjoints

$$\begin{array}{ccc} \mathbf{Mon}^{\text{op}} & \xrightarrow{U} & \mathbf{Set}^{\text{op}} \\ y^- \downarrow & & \downarrow y^- \\ \mathbf{Cat}^\sharp & \xrightarrow{U} & \mathbf{Poly} \end{array}$$

where the functors denoted  $U$  are forgetful functors.

*Proof.* Using the fully faithful functor  $y^-: \mathbf{Mon}^{\text{op}} \hookrightarrow \mathbf{Cat}^\sharp$  from Proposition 6.78, it is easy to check that the above diagram commutes.

The free-forgetful adjunction  $\mathbf{Set} \hookleftarrow \mathbf{Mon}$  gives an opposite adjunction  $\mathbf{Set}^{\text{op}} \hookleftarrow \mathbf{Mon}^{\text{op}}$ , where  $U$  is now left adjoint. We saw that  $y^-: \mathbf{Set}^{\text{op}} \rightarrow \mathbf{Poly}$  is a left adjoint in Proposition 4.12, that  $U: \mathbf{Cat}^\sharp \rightarrow \mathbf{Poly}$  is a left adjoint in Theorem 7.45, and that  $y^-: \mathbf{Mon} \rightarrow \mathbf{Cat}^\sharp$  is a left adjoint in Theorem 7.59.  $\square$

## 7.2 More categorical properties of $\mathbf{Cat}^\sharp$

Many of the properties of  $\mathbf{Poly}$  we covered in Chapters 2 and 4 have analogues in  $\mathbf{Cat}^\sharp$ ; we review these here.

### 7.2.1 Other special comonoids and adjunctions

We begin by highlighting a few other adjunctions involving  $\mathbf{Cat}^\sharp$ , as well as the special comonoids in  $\mathbf{Cat}^\sharp$  they provide.

**Proposition 7.61.** The functor  $y^-$  from Proposition 7.60 factors through an isomorphism of categories

$$\mathbf{Cat}_{\text{rep}}^{\#} \cong \mathbf{Mon}^{\text{op}},$$

where  $\mathbf{Mon}$  is the category of monoids and monoid homomorphisms and  $\mathbf{Cat}_{\text{rep}}^{\#}$  is the full subcategory of  $\mathbf{Cat}^{\#}$  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 cofunctors between monoids are dual—opposite—to morphisms between monoids.

A cofunctor  $f: y^M \rightarrow y^N$  involves a single function  $f^{\#}: N \rightarrow M$  that must satisfy a law coming from unitality and one coming from composition, as in Definition 6.49. The result can now be checked by hand, or seen formally as follows. Each object in the two diagrams of (6.49) is representable by Exercise 5.9. The Yoneda embedding  $\mathbf{Set}^{\text{op}} \rightarrow \mathbf{Poly}$  is fully faithful, so these two diagrams are equivalent to the unit and composition diagrams for monoid homomorphisms.  $\square$

*Exercise 7.62.* Let  $\mathbf{Cat}_{\text{lin}}^{\#}$  be the full subcategory of  $\mathbf{Cat}^{\#}$  consisting of categories with linear carriers  $Sy$  for some  $S \in \mathbf{Set}$ . Show that there is an isomorphism of categories

$$\mathbf{Cat}_{\text{lin}}^{\#} \cong \mathbf{Set}. \quad \diamond$$

**Proposition 7.63** (Discrete categories). The inclusion  $\mathbf{Cat}_{\text{lin}}^{\#} \rightarrow \mathbf{Cat}^{\#}$  has a left adjoint sending each  $(\mathfrak{c}, \epsilon, \delta) \in \mathbf{Cat}^{\#}$  to the unique comonoid carried by  $(\mathfrak{c} \triangleleft 1)y$  in  $\mathbf{Cat}_{\text{lin}}^{\#}$ .

*Proof.* We need to show that for any comonoid  $(\mathfrak{c}, \epsilon, \delta)$  and set  $A$ , we have a natural isomorphism

$$\mathbf{Cat}^{\#}(\mathfrak{c}, Ay) \cong \mathbf{Cat}^{\#}((\mathfrak{c} \triangleleft 1)y, Ay)$$

But every morphism in  $Ay$  is an identity, so the result follows from the fact that every cofunctor must pass identities back to identities.  $\square$

## 7.2.2 Vertical-cartesian factorization of cofunctors

A cofunctor is called *cartesian* if the underlying lens  $f: \mathfrak{c} \rightarrow \mathfrak{d}$  is cartesian (i.e. for each position  $i \in \mathfrak{c}(1)$ , the map  $f_i^{\#}: \mathfrak{d}[f_1(i)] \rightarrow \mathfrak{c}[i]$  is an isomorphism).

**Proposition 7.64.** Every cofunctor  $f: \mathcal{C} \rightarrow \mathcal{D}$  factors as a vertical morphism followed by a cartesian morphism

$$\mathcal{C} \xrightarrow{\text{vert}} \mathcal{C}' \xrightarrow{\text{cart}} \mathcal{D}.$$

*Proof.* A cofunctor  $\mathcal{C} \rightarrow \mathcal{D}$  is a map of polynomials  $\mathfrak{c} \rightarrow \mathfrak{d}$  satisfying some properties, and any map of polynomials  $f: \mathfrak{c} \rightarrow \mathfrak{d}$  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  $g_1: \mathfrak{c}(1) \rightarrow \mathfrak{c}'(1)$  is identity (rather than merely isomorphism) on positions and similarly that for each  $i \in \mathfrak{c}$  the map  $h_i^\# : \mathfrak{c}'[i] \rightarrow \mathfrak{d}[h_1(i)]$  is identity (rather than merely isomorphism) on directions.

It suffices to show that the intermediate object  $\mathfrak{c}'$  can be endowed with the structure of a category such that  $g$  and  $h$  are cofunctors. Given an object  $i \in \mathfrak{c}'(1)$ , assign its identity to be the identity on  $h_1(i) = f(i)$ ; then both  $g$  and  $h$  preserve identities because  $f$  does. Given an emanating morphism  $m \in \mathfrak{c}'[i] = \mathfrak{d}[f(i)]$ , assign its codomain to be  $\text{cod}(m) := \text{cod}(f_i^\#(m))$ , and given an emanating morphism  $m' \in \mathfrak{c}'[\text{cod}(m)]$ , assign the composite  $m \circ m'$  in  $\mathfrak{c}'$  to be  $m \circ m'$  in  $\mathfrak{d}$ . In Exercise 7.65 we will check that with these definitions,  $\mathfrak{c}'$  is a category and both  $g$  and  $h$  are cofunctors.  $\square$

*Exercise 7.65.* We will complete the proof of Proposition 7.64, 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.  $\diamond$

**Proposition 7.66.** The wide<sup>a</sup> subcategory of cartesian cofunctors in  $\mathbf{Cat}^\#$  is isomorphic to the wide subcategory of discrete opfibrations in  $\mathbf{Cat}$ .

<sup>a</sup> A subcategory  $\mathbf{D}$  of a category  $\mathbf{C}$  is *wide* if every object of  $\mathbf{C}$  is in  $\mathbf{D}$ .

*Proof.* Suppose that  $\mathcal{C}$  and  $\mathcal{D}$  are categories. Both a functor and a cofunctor between them involve a map on objects, say  $f: \text{Ob } \mathcal{C} \rightarrow \text{Ob } \mathcal{D}$ . For any object  $c \in \text{Ob } \mathcal{C}$ , a functor gives a function, say  $f_\# : \mathcal{C}[c] \rightarrow \mathcal{D}[f(c)]$  whereas a cofunctor gives a function  $f^\# : \mathcal{D}[f(c)] \rightarrow \mathcal{C}[c]$ . The cofunctor is cartesian iff  $f^\#$  is an iso, and the functor is a discrete opfibration iff  $f_\#$  is an iso. We thus transform our functor into a cofunctor (or vice versa) by taking the inverse function on morphisms. It is easy to check that this inverse appropriately preserves identities, codomains, and compositions.  $\square$

**Proposition 7.67.** The wide subcategory of vertical maps in  $\mathbf{Cat}^\#$  is isomorphic to the opposite of the wide subcategory bijective-on-objects maps in  $\mathbf{Cat}$ :

$$\mathbf{Cat}_{\text{vert}}^\# \cong (\mathbf{Cat}_{\text{boo}})^{\text{op}}.$$

*Proof.* Let  $\mathcal{C}$  and  $\mathcal{D}$  be categories. Given a vertical cofunctor  $F: \mathcal{C} \rightarrow \mathcal{D}$ , we have a bijection  $F_1: \text{Ob } \mathcal{C} \rightarrow \text{Ob } \mathcal{D}$ ; let  $G_1$  be its inverse. We define a functor  $G: \mathcal{D} \rightarrow \mathcal{C}$  on objects by  $G_1$  and, for any  $f: d \rightarrow d'$  in  $\mathcal{D}$  we define  $G(f) := F_{G_1(d)}^\#$ . It has the correct codomain:  $\text{cod}(G(f)) = G_1(F_1(\text{cod}(G(f)))) = G_1(\text{cod } f)$ . And it sends identities and compositions to identities and compositions by the laws of cofunctors.

The construction of a vertical cofunctor from a bijective-on-objects functor is analogous, and it is easy to check that the two constructions are inverses.  $\square$

*Exercise 7.68.* Let  $S$  be a set and consider the state category  $\mathcal{S} := (Sy^S, \epsilon, \delta)$ . Use Proposition 7.67 to show that categories  $\mathcal{C}$  equipped with a vertical cofunctor  $\mathcal{S} \rightarrow \mathcal{C}$  can be identified with categories whose set of objects has been identified with  $S$ .  $\diamond$

*Exercise 7.69.* Consider the categories  $\mathcal{C} := \boxed{\bullet \rightrightarrows \bullet}$  and  $\mathcal{D} := \boxed{\bullet \rightarrow \bullet}$ . There is a unique bijective-on-objects (boo) functor  $F: \mathcal{C} \rightarrow \mathcal{D}$  and two boo functors  $G_1, G_2: \mathcal{D} \rightarrow \mathcal{C}$ . These have corresponding cofunctors going the other way.

1. Write down the morphism  $\mathfrak{d} \rightarrow \mathfrak{c}$  of carriers corresponding to  $F$ .
2. Write down the morphism  $\mathfrak{c} \rightarrow \mathfrak{d}$  of carriers corresponding to either  $G_1$  or  $G_2$ .  $\diamond$

We record the following proposition here; it will be useful in ??.

**Proposition 7.70.** If  $\varphi: p \rightarrow q$  is a cartesian lens, then  $\mathcal{T}_\varphi: \mathcal{T}_p \rightarrow \mathcal{T}_q$  is a cartesian cofunctor: that is, for each  $t \in \text{tree}_p$ , the on-morphisms function

$$(\mathcal{T}_\varphi)_t^\#: \mathcal{T}_q[\mathcal{T}_\varphi t] \xrightarrow{\cong} \mathcal{T}_p[t]$$

is a bijection.

*Proof.* Given  $\varphi: p \rightarrow q$  cartesian, each tree  $T \in \text{tree}_p$  is sent to a tree in  $\text{tree}_q$  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.  $\square$

### 7.2.3 Limits and colimits of comonoids

We saw in Theorems 4.32 and 4.42 that **Poly** has all small limits and colimits. It turns out that **Cat**<sup>#</sup> has all small limits and colimits as well. We start by discussing colimits in **Cat**<sup>#</sup>, as they are somewhat easier to get a handle on.

**Colimits in  $\mathbf{Cat}^\#$** 

It is a consequence of the forgetful-cofree adjunction from Theorem 7.45 that  $\mathbf{Cat}^\#$  inherits all the colimits from  $\mathbf{Poly}$ . As these results are somewhat technical, relying on a property of functors known as *comonadicity*, we defer their proofs to references.

**Proposition 7.71** (Porst). The forgetful functor  $\mathbf{Cat}^\# \rightarrow \mathbf{Poly}$  is comonadic.

*Proof.* The fact that a forgetful functor  $\mathbf{Cat}^\# \cong \mathbf{Comon}(\mathbf{Poly}) \rightarrow \mathbf{Poly}$  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 7.45.  $\square$

**Corollary 7.72.** The category  $\mathbf{Cat}^\#$  has all small colimits. They are created by the forgetful functor  $\mathbf{Cat}^\# \rightarrow \mathbf{Poly}$ .

*Proof.* A comonadic functor creates all colimits that exist in its codomain (see [nLa18]), and by Theorem 4.42, the category  $\mathbf{Poly}$  has all small colimits.  $\square$

**Example 7.73** (Coproducts in  $\mathbf{Cat}^\#$ ). Probably the most familiar kind of colimit in  $\mathbf{Cat}^\#$  is the coproduct, as Corollary 7.72 tells us that it agrees with the usual coproduct from  $\mathbf{Cat}$ . Here’s why.

For concreteness, let  $I$  be a set and  $(\mathcal{C}_i)_{i \in I}$  be categories with carriers  $(c_i)_{i \in I}$ . Then the coproduct of  $(\mathcal{C}_i)_{i \in I}$  in  $\mathbf{Cat}$  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

$$\mathrm{Ob} \left( \sum_{i \in I} \mathcal{C}_i \right) \cong \sum_{i \in I} \mathrm{Ob} \mathcal{C}_i = \sum_{i \in I} c_i(1) \cong \left( \sum_{i \in I} c_i \right) (1),$$

and whose morphisms out of each  $c \in \mathrm{Ob} \mathcal{C}_j \subseteq \mathrm{Ob} \sum_{i \in I} \mathcal{C}_i$  are just the morphisms out of  $c$  in the summand  $\mathcal{C}_j$ , so

$$\left( \sum_{i \in I} \mathcal{C}_i \right) [c] \cong \mathcal{C}_j[c] = c_j[c] \cong \left( \sum_{i \in I} c_i \right) [c]$$

Hence  $\sum_{i \in I} \mathcal{C}_i$  is carried by the coproduct of polynomials  $\sum_{i \in I} c_i$ .

It remains to show that  $\sum_{i \in I} \mathcal{C}_i$  is also the coproduct of  $(\mathcal{C}_i)_{i \in I}$  in  $\mathbf{Cat}^\#$ . We already know from Corollary 7.72 that it has the right carrier: the coproduct of carriers  $\sum_{i \in I} c_i$ . It also has the right morphisms, for any object  $(i, x)$  with  $x \in c_i(1)$ , the set of emanating morphisms is—and should be— $c_i[x]$ .

*Exercise 7.74.* 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 7.72.  $\diamond$

*Exercise 7.75.* Given a comonoid  $(c, \epsilon, \delta) \in \mathbf{Cat}^\sharp$ , show that there is an induced comonoid structure on the polynomial  $2c$ .  $\diamond$

### 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 7.76.** The category  $\mathbf{Cat}^\sharp$  has all small limits.

*Proof.* By Theorem 7.45, the forgetful functor  $U: \mathbf{Cat}^\sharp \rightarrow \mathbf{Poly}$  has a right adjoint, and by Theorem 4.32,  $\mathbf{Poly}$  itself has all small limits. Furthermore equalizers in  $\mathbf{Poly}$  are connected limits, so by Theorem 5.86, they are preserved by  $\triangleleft$  on either side. Then the result follows from [Por19, Fact 3.4].  $\square$

### 7.2.4 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  $\mathbf{Poly}$ .

**Proposition 7.77.** The parallel product  $(y, \otimes)$  on  $\mathbf{Poly}$  extends to a monoidal structure  $(y, \otimes)$  on  $\mathbf{Cat}^\sharp$ , such that the forgetful functor  $U: \mathbf{Cat}^\sharp \rightarrow \mathbf{Poly}$  is strong monoidal with respect to  $\otimes$ . The parallel product of two categories is their product in  $\mathbf{Cat}$ .

*Proof.* Let  $\mathcal{C}, \mathcal{D} \in \mathbf{Cat}^\sharp$  be categories corresponding to comonoids  $(c, \epsilon_c, \delta_c)$  and  $(d, \epsilon_d, \delta_d)$ .

The carrier of  $\mathcal{C} \otimes \mathcal{D}$  is defined to be  $c \otimes d$ . 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  $\epsilon_{\mathcal{C} \otimes \mathcal{D}}: c \otimes d \rightarrow y$  as

$$c \otimes d \xrightarrow{\epsilon_c \otimes \epsilon_d} y \otimes y \cong y.$$

This says that the identity at  $(c, d)$  is the pair of identities.

We define  $\delta_{\mathcal{C} \otimes \mathcal{D}}: (c \otimes d) \rightarrow (c \otimes d) \triangleleft (c \otimes d)$  using the duoidal property:

$$c \otimes d \xrightarrow{\delta_c \otimes \delta_d} (c \triangleleft c) \otimes (d \triangleleft d) \rightarrow (c \otimes d) \triangleleft (c \otimes d).$$

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 7.78.  $\square$

*Exercise 7.78.* We complete the proof of Proposition 7.77.

1. Show that  $(\mathfrak{c} \otimes \mathfrak{d}, \epsilon_{\mathfrak{c} \otimes \mathfrak{d}}, \delta_{\mathfrak{c} \otimes \mathfrak{d}})$ , as described in Proposition 7.77, forms a comonoid.
2. Check that the construction  $(\mathcal{C}, \mathcal{D}) \mapsto \mathcal{C} \otimes \mathcal{D}$  is functorial in  $\mathcal{C}, \mathcal{D} \in \mathbf{Cat}^\sharp$ .  $\diamond$

The cofree construction works nicely with this monoidal product.

**Proposition 7.79.** The cofree functor  $p \mapsto \mathcal{T}_p$  is lax monoidal; in particular there is a map of polynomials  $y \rightarrow \mathfrak{t}_y$ , and for any  $p, q \in \mathbf{Poly}$  there is a natural map

$$\mathfrak{t}_p \otimes \mathfrak{t}_q \rightarrow \mathfrak{t}_{p \otimes q}.$$

satisfying the usual conditions.

*Proof.* By Proposition 7.77, the left adjoint  $U: \mathbf{Cat}^\sharp \rightarrow \mathbf{Poly}$  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.  $\square$

- Exercise 7.80.*
1. What polynomial is  $\mathfrak{t}_y$ ?
  2. What is the map  $y \rightarrow \mathfrak{t}_y$  from Proposition 7.79?
  3. Explain in words how to think about the map  $\mathfrak{t}_p \otimes \mathfrak{t}_q \rightarrow \mathfrak{t}_{p \otimes q}$  from Proposition 7.79, for arbitrary  $p, q \in \mathbf{Poly}$ .  $\diamond$

## 7.3 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 category-theoretic 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.

### 7.3.1 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 7.81** (Left comodule). In a monoidal category  $(\mathbf{C}, y, \triangleleft)$ , let  $\mathcal{C} = (c, \epsilon, \delta)$  be a comonoid. A *left  $\mathcal{C}$ -comodule* is

- an object  $m \in \mathbf{C}$ , called the *carrier*, equipped with
- a morphism  $c \triangleleft m \xleftarrow{\lambda} m$  called the *left coaction*,

such that the following diagrams, collectively known as the *left comodule laws*, commute:

$$\begin{array}{ccc}
 c \triangleleft m & \xleftarrow{\lambda} & m \\
 \epsilon \triangleleft m \downarrow & & \downarrow \lambda \\
 m & \xleftarrow{\quad} & m
 \end{array}
 \qquad
 \begin{array}{ccc}
 c \triangleleft m & \xleftarrow{\lambda} & m \\
 \delta \triangleleft m \downarrow & & \downarrow \lambda \\
 c \triangleleft c \triangleleft m & \xleftarrow{c \triangleleft \lambda} & c \triangleleft m
 \end{array}
 \tag{7.82}$$

When referring to a left  $\mathcal{C}$ -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  $\mathcal{C}$ -comodules  $m$  and  $n$  is a morphism  $\alpha: m \rightarrow n$  such that the following diagram commutes:

$$\begin{array}{ccc}
 c \triangleleft m & \xleftarrow{\quad} & m \\
 c \triangleleft \alpha \downarrow & & \downarrow \alpha \\
 c \triangleleft n & \xleftarrow{\quad} & n
 \end{array}$$

Here the top and bottom morphisms are the left coactions of  $m$  and  $n$ .

*Exercise 7.83.* Show that the category of  $\mathcal{C}$ -coalgebras from Definition 6.95 is exactly the category of *constant* left  $\mathcal{C}$ -comodules—i.e. the full subcategory of the category of left  $\mathcal{C}$ -comodules spanned by those left  $\mathcal{C}$ -comodules whose carriers are constant polynomials.  $\diamond$

**Definition 7.84** (Right comodule). In a monoidal category  $(\mathbf{C}, y, \triangleleft)$ , let  $\mathcal{D} = (d, \epsilon, \delta)$  be a comonoid. A *right  $\mathcal{D}$ -comodule* is

- an object  $m \in \mathbf{C}$ , called the *carrier*, equipped with
- a morphism  $m \xrightarrow{\rho} m \triangleleft d$  called the *right coaction*,

such that the following diagrams, collectively known as the *right comodule laws*, commute:

$$\begin{array}{ccc}
 m & \xrightarrow{\rho} & m \triangleleft d \\
 \downarrow \rho & & \downarrow m \triangleleft \epsilon \\
 m & \xrightarrow{\quad} & m
 \end{array}
 \qquad
 \begin{array}{ccc}
 m & \xrightarrow{\rho} & m \triangleleft d \\
 \rho \downarrow & & \downarrow m \triangleleft \delta \\
 m \triangleleft d & \xrightarrow{\rho \triangleleft d} & m \triangleleft d \triangleleft d
 \end{array}
 \tag{7.85}$$

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  $\alpha: m \rightarrow n$  such that the



following diagram commutes:

$$\begin{array}{ccc} m & \longrightarrow & m \triangleleft \mathfrak{d} \\ \alpha \downarrow & & \downarrow \alpha \triangleleft \mathfrak{d} \\ n & \longrightarrow & n \triangleleft \mathfrak{d} \end{array}$$

Here the top and bottom morphisms are the right coactions of  $m$  and  $n$ .

- Exercise 7.86.* 1. Draw the equations of (7.82) using polyboxes.  
2. Draw the equations of (7.85) using polyboxes.  $\diamond$

- Exercise 7.87.* Recall from Exercise 6.63 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.  $\diamond$

We can characterize left comodules as something far more familiar.

**Proposition 7.88.** Let  $\mathcal{C}$  be a category. The category of left  $\mathcal{C}$ -comodules is equivalent to the category of functors  $\mathcal{C} \rightarrow \mathbf{Poly}$ .

*Sketch of proof.* Let  $\mathfrak{c}$  be the carrier of  $\mathcal{C}$ . Given a left  $\mathcal{C}$ -comodule  $m \rightarrow \mathfrak{c} \triangleleft m$ , we saw in Exercise 7.86 that there is an induced function  $|-|: m(1) \rightarrow \mathfrak{c}(1)$ , so to each object  $i \in \text{Ob}(\mathcal{C})$  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  $p_i^m \in \mathbf{Poly}$  is easily seen to be functorial in the comodule  $m$ . Moreover, given a morphism  $f: i \rightarrow i'$  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.  $a.f \in P_{i'}^m = p_{i'}^m(1)$ . Similarly for each  $x \in m[a.f] = p_{i'}^m[a.f]$  we have  $\lambda_a^\#(f, x) \in m[a] = p_i^m[a]$ , and thus we have a map of polynomials  $p_i^m \rightarrow p_{i'}^m$ . In this way we have obtained we obtain a functor  $p_-^m: \mathcal{C} \rightarrow \mathbf{Poly}$ , and this functor is itself functorial in  $m$ .

We leave the proof that this construction has an inverse to the reader.  $\square$

Similarly, a right comodule is not an altogether foreign concept.

**Proposition 7.89.** Let  $\mathcal{D}$  be a category. A right  $\mathcal{D}$ -comodule  $m$  can be identified (up to isomorphism) with a functor  $\mathcal{D} \rightarrow \mathbf{Set}^{m(1)}$ .

*Sketch of proof.* Let  $\mathfrak{d}$  be the carrier of  $\mathcal{D}$ . Given a right  $\mathcal{D}$ -comodule  $m \rightarrow m \triangleleft \mathfrak{d}$  we need a functor  $F: \mathcal{D} \times m(1) \rightarrow \mathbf{Set}$ . We again rely heavily on Exercise 7.86. Given  $j \in \mathcal{D}$  and  $a \in m(1)$ , define

$$F(j, d) := \{x \in m[a] \mid |a.f| = j\}.$$

Given  $g: j \rightarrow j'$  we get  $\rho_a^\#(x, g) \in m[a]$  with  $|a.\rho_a^\#(x, g)| = j'$ , and thus  $g$  induces a function  $F(g, d): F(j, d) \rightarrow F(j', d)$ . Again by laws from Exercise 7.86 these are functorial in  $g$ , completing the proof sketch.  $\square$

**Proposition 7.90.** Let  $\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$  be a comonoid in **Poly**. For any set  $G$ , the polynomial  $y^G \triangleleft \mathfrak{c}$  has a natural right  $\mathcal{C}$ -comodule structure.

*Proof.* We use the map  $(y^G \triangleleft \delta): (y^G \triangleleft \mathfrak{c}) \rightarrow (y^G \triangleleft \mathfrak{c} \triangleleft \mathfrak{c})$ . It satisfies the unitality and associativity laws because  $\mathfrak{c}$  does.  $\square$

We can think of elements of  $G$  as “generators”. Then if  $i': G \rightarrow \mathfrak{c} \triangleleft 1$  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 7.91.** Functions  $i': G \rightarrow \mathfrak{c} \triangleleft 1$  are in bijection with positions  $i \in y^G \triangleleft \mathfrak{c} \triangleleft 1$ . Let  $m := i^*(y^G \triangleleft \mathfrak{c})$  and let  $\rho_i$  be the induced right  $\mathcal{C}$ -comodule structure from ???. Then  $\rho_i$  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  $\rho_i$  is a representable right  $\mathcal{C}$ -comodule, and we can identify it with a  $\mathcal{C}$ -set by Theorem 7.100. The elements of this  $\mathcal{C}$ -set are pairs  $(g, f)$ , where  $g \in G$  is a generator and  $f: i'(g) \rightarrow \text{cod}(f)$  is a morphism in  $\mathcal{C}$  emanating from  $i'(g)$ . It is easy to see that the comodule structure induced by Proposition 7.90 is indeed the free one.  $\square$

*Exercise 7.92.* Let  $\mathcal{C}$  be a category and  $i \in \mathcal{C}$  an object.

1. Consider  $i$  as a map  $y \rightarrow \mathfrak{c}$ . Show that the vertical-cartesian factorization of this map is  $y \rightarrow y^{\mathfrak{c}[i]} \xrightarrow{\varphi} \mathfrak{c}$ .
2. Use Proposition 5.94 to show that  $y^{\mathfrak{c}[i]} \triangleleft \mathfrak{c} \rightarrow \mathfrak{c} \triangleleft \mathfrak{c}$  is cartesian.
3. Show that there is a commutative square

$$\begin{array}{ccc} y^{\mathfrak{c}[i]} & \xrightarrow{\delta^i} & y^{\mathfrak{c}[i]} \triangleleft \mathfrak{c} \\ \varphi \downarrow & \lrcorner & \downarrow \text{cart} \\ \mathfrak{c} & \xrightarrow{\delta} & \mathfrak{c} \triangleleft \mathfrak{c} \end{array}$$

4. Show that this square is a pullback, as indicated.

5. Show that  $\delta^i$  makes  $y^{c[i]}$  a right  $\mathcal{C}$ -comodule.  $\diamond$

The map  $\delta^i$  can be seen as the restriction of  $\delta: c \rightarrow c \triangleleft c$  to a single starting position.

We can extend this to a functor  $\mathcal{C} \rightarrow {}_y\mathbf{Mod}_{\mathcal{C}}$  that sends the object  $i$  to  $y^{c[i]}$ . Given a morphism  $f: i \rightarrow i'$  in  $\mathcal{C}$ , we get a function  $c[i'] \rightarrow c[i]$  given by composition with  $f$ , and hence a map of polynomials  $y^{c[f]}: y^{c[i]} \rightarrow y^{c[i']}$ .

*Exercise 7.93.* 1. Show that  $y^{c[f]}$  is a map of right  $\mathcal{C}$ -comodules.

2. Show that the construction  $y^{c[f]}$  is functorial in  $f$ .  $\diamond$

**Proposition 7.94.** Let  $c$  be a comonoid. For any set  $I$  and right  $c$ -comodules  $(m_i)_{i \in I}$ , the coproduct  $m := \sum_{i \in I} m_i$  has a natural right-comodule structure. Moreover, each representable summand in the carrier  $m$  of a right  $c$ -comodule is itself a right- $c$  comodule and  $m$  is their sum.

*Sketch of proof.* This follows from the fact in Proposition 5.53 that  $- \triangleleft c$  commutes with coproduct, i.e.  $(\sum_{i \in I} m_i) \triangleleft c \cong \sum_{i \in I} (m_i \triangleleft c)$ .  $\square$

**Proposition 7.95.** If  $m \in \mathbf{Poly}$  is equipped with both a right  $\mathcal{C}$ -comodule and a right  $\mathcal{D}$ -comodule structure, we can naturally equip  $m$  with a  $(\mathcal{C} \times \mathcal{D})$ -comodule structure.

*Proof.* It suffices by Proposition 7.94 to assume that  $m = y^M$  is representable. But a right  $\mathcal{C}$ -comodule with carrier  $y^M$  can be identified with a cofunctor  $My^M \rightarrow \mathcal{C}$ .

Thus if  $y^M$  is both a right- $\mathcal{C}$  comodule and a right- $\mathcal{D}$  comodule, then we have comonoid morphisms  $\mathcal{C} \leftarrow My^M \rightarrow \mathcal{D}$ . This induces a unique comonoid morphism  $My^M \rightarrow (\mathcal{C} \times \mathcal{D})$  to the product, and we identify it with a right- $(\mathcal{C} \times \mathcal{D})$  comodule on  $y^M$ .  $\square$

### 7.3.2 Bicomodules

We take special note of any object that is both a left comodule and a right comodule in compatible ways.

**Definition 7.96 (Bicomodule).** In a monoidal category  $(\mathbf{C}, y, \triangleleft)$ , let  $\mathcal{C}$  and  $\mathcal{D}$  be comonoids with carriers  $c$  and  $d$ . 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  $c \triangleleft m \xleftarrow{\lambda} m$ , and
3. a right  $\mathcal{D}$ -comodule, with right coaction  $m \xrightarrow{\rho} m \triangleleft d$ ,

such that the following diagram, known as the *coherence law*, commutes:

$$\begin{array}{ccc}
 & m & \\
 \lambda \swarrow & & \searrow \rho \\
 c \triangleleft m & & m \triangleleft d \\
 \searrow \rho & & \swarrow \lambda \\
 c \triangleleft m \triangleleft d & & c \triangleleft m \triangleleft d
 \end{array} \quad (7.97)$$

We may denote such a bicomodule by  $C \xleftarrow{m} D$  when its left and right coactions have yet to be specified or may be inferred from context—or even by  $c \xleftarrow{m} d$  when the comonoid structures on  $c$  and  $d$  can be inferred from context as well.<sup>a</sup>

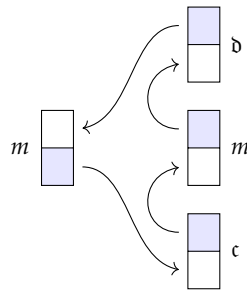
A *morphism* of  $(C, D)$ -bicomodules is one that is a morphism of left  $C$ -comodules and a morphism of right  $D$ -comodules.

<sup>a</sup>The notation  $c \xleftarrow{m} d$  has a mnemonic advantage, as each  $\triangleleft$  goes in the correct direction:  $c \triangleleft m \leftarrow m$  and  $m \rightarrow m \triangleleft d$ . But it also looks like an arrow going backward from  $d$  to  $c$ , which will turn out to have a semantic advantage as well

We draw the commutativity of (7.97) using polyboxes.

$$\text{Polybox diagram showing the commutativity of the coherence law (7.97).} \quad (7.98)$$

This polybox equation implies that we can unambiguously write



for any bicomodule  $c \xleftarrow{m} d$ .

**Exercise 7.99.** Let  $C = (c, \epsilon, \delta)$  be a category. Recall from Exercise 6.63 that  $y$  has a unique category structure.

1. Show that a left  $C$ -module is the same thing as a  $(C, y)$ -bimodule.
2. Show that a right  $C$ -module is the same thing as a  $(y, C)$ -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  $\mathbf{Poly} \cong {}_y\mathbf{Mod}_y$ .  $\diamond$

### 7.3.3 More equivalences

In ??, we saw that a cofunctor 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  $\mathcal{C}$ . Then in ??, we showed that cartesian cofunctors 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.

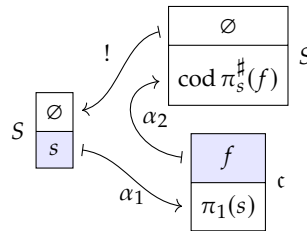
**Theorem 7.100.** Given a polynomial comonoid  $\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$ , the following comprise equivalent categories:

1. functors  $\mathcal{C} \rightarrow \mathbf{Set}$ ;
2. discrete opfibrations over  $\mathcal{C}$ ;
3. cartesian cofunctors to  $\mathcal{C}$ ;
4.  $\mathcal{C}$ -coalgebras (sets with a coaction by  $\mathcal{C}$ );
5. constant left  $\mathcal{C}$ -comodules;
6.  $(\mathcal{C}, 0)$ -bicomodules;
7. linear left  $\mathcal{C}$ -comodules; and
8. representable right  $\mathcal{C}$ -comodules (opposite).

In fact, all but the first comprise isomorphic categories; and up to isomorphism, any one of these can be identified with a cofunctor from a state category to  $\mathcal{C}$ .

*Proof.*  $1 \simeq 2 \cong 3$ : This was shown in Proposition 6.107.

$3 \cong 4$ : Given a cartesian cofunctor  $(\pi_1, \pi^\#): \mathcal{S} \rightarrow \mathcal{C}$ , let  $S := \text{Ob}(\mathcal{S})$  and define a  $\mathfrak{c}$ -coalgebra structure  $\alpha: S \rightarrow \mathfrak{c} \triangleleft S$  on an object  $s \in \text{Ob}(\mathcal{S})$  and an emanating morphism  $f: \pi_1(s) \rightarrow c'$  in  $\mathcal{C}$  by



We check that this is indeed a coalgebra using properties of cofunctors. For identities in  $\mathcal{C}$ , we have

$$\begin{aligned} \alpha_2(s, \text{id}_{\pi_1(s)}) &= \text{cod } \pi_s^\#(\text{id}_{\pi_1(s)}) \\ &= \text{cod id}_s = s \end{aligned}$$

and for compositions in  $\mathcal{C}$  we have

$$\alpha_2(s, f \circ g) = \text{cod} \left( \pi_s^\#(f \circ g) \right)$$

$$\begin{aligned}
&= \text{cod} \left( \pi_s^\#(f) \circ \pi_{\text{cod } \pi_s^\#(f)}^\# g \right) \\
&= \text{cod} \left( \pi_{\text{cod } \pi_s^\#(f)}^\# g \right) \\
&= \alpha_2(\alpha_2(s, f), g).
\end{aligned}$$

Going backward, if we're given a coalgebra  $\alpha: S \rightarrow c \triangleleft S$ , we obtain a function  $\alpha_1: S \rightarrow c \triangleleft 1$  and we define  $s := \alpha_1^* c$  and the cartesian map  $\varphi := (\alpha_1, \text{id}): s \rightarrow c$  to be the base change from Proposition 4.71. We need to show  $s$  has a comonoid structure  $(s, \epsilon, \delta)$  and that  $\varphi$  is a cofunctor. We simply define the eraser  $\epsilon: s \rightarrow y$  using  $\alpha_1$  and the eraser on  $c$ :

$$\begin{array}{ccc}
\begin{array}{|c|} \hline \text{id}_{\alpha_1(s)} \\ \hline s \\ \hline \end{array} & \xrightarrow{\alpha_1} & \begin{array}{|c|} \hline \text{id}_{\alpha_1 s} \\ \hline \alpha_1(s) \\ \hline \end{array} \\
s & & c
\end{array}$$

$\epsilon_c$

We give the duplicator  $\delta: s \rightarrow s \triangleleft s$  using  $\alpha_2$  for the codomain, and using the composite  $\circ$  from  $c$ :

$$\begin{array}{ccc}
& & \begin{array}{|c|} \hline g \\ \hline \alpha_2(s, f) \\ \hline \end{array} s \\
& \swarrow \text{com} & \uparrow \\
\begin{array}{|c|} \hline f \circ g \\ \hline s \\ \hline \end{array} s & & \begin{array}{|c|} \hline f \\ \hline s \\ \hline \end{array} s \\
& \searrow \text{cod} & \downarrow
\end{array}$$

In Exercise 7.101 we check that  $(s, \epsilon, \delta)$  really is a comonoid, that  $(\alpha_1, \text{id}): s \rightarrow c$  is a cofunctor, that the roundtrips between cartesian cofunctors and coalgebras are identities, and that these assignments are functorial.

- 4  $\cong$  5: This is straightforward and was mentioned in Definition 6.95.
- 5  $\cong$  6: A right 0-comodule is in particular a polynomial  $m \in \mathbf{Poly}$  and a map  $\rho: m \rightarrow m \triangleleft 0$  such that  $(m \triangleleft \epsilon) \circ \rho = \text{id}_m$ . This implies  $\rho$  is monic, which itself implies by Proposition 4.17 that  $m$  must be constant since  $m \triangleleft 0$  is constant. This makes  $m \triangleleft \epsilon$  the identity, at which point  $\rho$  must also be the identity. Conversely, for any set  $M$ , the corresponding constant polynomial is easily seen to make the diagrams in (??) commute.
- 5  $\cong$  7: By the adjunction in Proposition 1.18 and the fully faithful inclusion  $\mathbf{Set} \rightarrow \mathbf{Poly}$  of sets as constant polynomials, Proposition 4.2, we have isomorphisms

$$\mathbf{Poly}(Sy, c \triangleleft Sy) \cong \mathbf{Set}(S, c \triangleleft Sy \triangleleft 1) = \mathbf{Set}(S, c \triangleleft S) \cong \mathbf{Poly}(S, c \triangleleft S).$$

One checks easily that if  $Sy \rightarrow c \triangleleft Sy$  corresponds to  $S \rightarrow c \triangleleft S$  under the above isomorphism, then one is a left comodule if and only if the other is.

- 7  $\cong$  8: By (5.72) we have a natural isomorphism

$$\mathbf{Poly}(Sy, c \triangleleft Sy) \cong \mathbf{Poly}(y^S, y^S \triangleleft c).$$

In pictures,



The last claim was proven in Proposition 6.108.  $\square$

**Exercise 7.101.** Complete the proof of Theorem 7.100 (3  $\cong$  4) by proving the following.

1. Show that  $(s, \epsilon, \delta)$  really is a comonoid.
2. Show that  $(\alpha_1, \text{id}): s \rightarrow c$  is a cofunctor.
3. Show that the roundtrips between cartesian cofunctors and coalgebras are identities.
4. Show that the assignment of a  $\mathcal{C}$ -coalgebra to a cartesian cofunctor over  $\mathcal{C}$  is functorial.
5. Show that the assignment of a cartesian cofunctor over  $\mathcal{C}$  to a  $\mathcal{C}$ -coalgebra is functorial.  $\diamond$

Let  $\mathcal{C}$  be a category. Under the above correspondence, the terminal functor  $\mathcal{C} \rightarrow \mathbf{Set}$  corresponds to the identity discrete opfibration  $\mathcal{C} \rightarrow \mathcal{C}$ , the identity cofunctor  $\mathcal{C} \rightarrow \mathcal{C}$ , a certain left  $\mathcal{C}$  comodule with carrier  $\mathcal{C}(1)y$  which we call the *canonical left  $\mathcal{C}$ -comodule*, a certain constant left  $\mathcal{C}$  comodule with carrier  $\mathcal{C}(1)$  which we call the *canonical  $(\mathcal{C}, 0)$ -bicomodule*, and a certain representable right  $\mathcal{C}$ -comodule with carrier  $y^{\mathcal{C}(1)}$  which we call the *canonical right  $\mathcal{C}$ -comodule*.

**Exercise 7.102.** For any object  $c \in \mathcal{C}$ , consider the representable functor  $\mathcal{C}(c, -): \mathcal{C} \rightarrow \mathbf{Set}$ . What does it correspond to as a

1. discrete opfibration over  $\mathcal{C}$ ?
2. cartesian cofunctor to  $\mathcal{C}$ ?
3. linear left  $\mathcal{C}$ -comodule?
4. constant left  $\mathcal{C}$ -comodule?
5.  $(\mathcal{C}, 0)$ -bicomodule?
6. representable right  $\mathcal{C}$ -comodule?
7. dynamical system with comonoid interface  $\mathcal{C}$ ?  $\diamond$

**Exercise 7.103.** We saw in Theorem 7.100 that the category  ${}_c\mathbf{Mod}_0$  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  ${}_0\mathbf{Mod}_{\mathcal{C}}$ ?

◇

### 7.3.4 Bicomodules are parametric right adjoints

There is an equivalent characterization of bicomodules over a pair of polynomial comonoids due to Richard Garner.

**Proposition 7.104** (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  $\mathbf{Set}^{\mathcal{D}} \rightarrow \mathbf{Set}^{\mathcal{C}}$ .
3. a connected limit-preserving functor  $\mathbf{Set}^{\mathcal{D}} \rightarrow \mathbf{Set}^{\mathcal{C}}$ .

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:

$$m(1) \in \mathbf{Set}^{\mathcal{D}} \quad \text{and} \quad m[i] \in \mathbf{Set}^{\mathcal{C}}$$

Before we knew about bicomodule structures, what we called positions and directions—and what we often think of as outputs and inputs of a system—were understood as each forming an ordinary set. In the presence of a 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 7.105** (Prafunctor). Let  $\mathcal{C}$  and  $\mathcal{D}$  be categories. A *prafunctor* (also called a *parametric right adjoint functor*)  $\mathbf{Set}^{\mathcal{C}} \rightarrow \mathbf{Set}^{\mathcal{D}}$  is one satisfying any of the conditions of Proposition 7.104.

### 7.3.5 Bicomodules in dynamics

We conclude with a peek at how bicomodules can model dynamical systems themselves.

*Example 7.106* (Cellular automata). In Example 3.68 and Exercise 3.69 we briefly discussed cellular automata; here we will discuss another way that cellular automata show up, this time in terms of bicomodules.

Suppose that  $\text{src}, \text{tgt}: A \rightrightarrows V$  is a graph, and consider the polynomial

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



so that positions are vertices and directions are emanating arrows. It carries a natural bicomodule structure

$$Vy \xleftarrow{g} Vy$$

where the right structure map uses  $\text{tgt}$ ; see Exercise 7.107 for details. A bicomodule

$$Vy \xleftarrow{T} 0$$

can be identified with a functor  $T: V \rightarrow \mathbf{Set}$ , 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

$$\begin{array}{c} Vy \xleftarrow{g} Vy \xleftarrow{T} 0 \\ \quad \downarrow \alpha \\ \quad T \end{array}$$

Indeed, for every vertex  $v \in V$  the map  $\alpha$  gives a function

$$\prod_{\text{src}(a)=v} T(\text{tgt}(a)) \xrightarrow{\alpha_v} T(v),$$

which we call the *update* function. In other words, given the current color at the target of each arrow emanating from  $v$ , the function  $\alpha_v$  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 7.107.* Let  $\text{src}, \text{tgt}: A \rightrightarrows V$  and  $g$  and  $T$  be as in Example 7.106.

1. Give the structure map  $\lambda: g \rightarrow Vy \triangleleft g$
2. Give the structure map  $\rho: g \rightarrow g \triangleleft Vy$ .
3. Give the set  $T$  and the structure map  $T \rightarrow Vy \triangleleft T$  corresponding to the functor  $V \rightarrow \mathbf{Set}$  that assigns 2 to each vertex.  $\diamond$

*Example 7.108* (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  $\alpha$  be the update function for a cellular automaton. As in Example 7.106, this is all given by a diagram

$$\begin{array}{c} Vy \xleftarrow{g} Vy \xleftarrow{T} 0 \\ \quad \downarrow \alpha \\ \quad T \end{array}$$

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

$$\begin{array}{ccc}
 & V & \\
 \swarrow & \Downarrow \sigma & \searrow \\
 Vy & & 0 \\
 \nwarrow & \Uparrow T & \nearrow
 \end{array} \quad (7.109)$$

Note that  $g$  is a profunctor, i.e. for each  $v \in V$  the summand  $g_v = y^{\text{src}^{-1}(v)}$  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

$$\begin{array}{ccccccc}
 & & & V & & & \\
 & & \swarrow & \parallel & \searrow & & \\
 Vy & \xleftarrow{g} & \cdots & \xleftarrow{g} & Vy & \xleftarrow{g} & Vy \xleftarrow{V} 0 \\
 \parallel & & & \parallel & & & \parallel \\
 Vy & \xleftarrow{g} & \cdots & \xleftarrow{g} & Vy & \xleftarrow{g} & Vy \xleftarrow{T} 0 \\
 \parallel & & & \parallel & & & \parallel \\
 Vy & \xleftarrow{g} & \cdots & \xleftarrow{g} & Vy & \xleftarrow{T} & 0 \\
 \parallel & & & \parallel & & & \parallel \\
 Vy & \xleftarrow{\quad} & & \downarrow \alpha \circ \cdots \circ \alpha & & & 0 \\
 & & & T & & & 
 \end{array}$$

*Exercise 7.110.* Explain why (7.109) models an initialization, i.e. a way to choose a starting color in each vertex.  $\diamond$

## 7.4 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

$$\mathbf{Cat}^\# \begin{array}{c} \xrightarrow{U} \\ \Rightarrow \\ \xleftarrow{\mathcal{T}_-} \end{array} \mathbf{Poly}$$

where  $\mathcal{T}_p$  is the cofree comonoid  $\mathcal{T}_p$  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  $\mathcal{T}_p$  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  $\mathbf{Cat}^\#$  including a formal proof of the fact that it has all limits and colimits. Colimits in  $\mathbf{Cat}^\#$  are created by, i.e. fit nicely with, colimits in  $\mathbf{Poly}$ , 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  $\boxed{\bullet \rightarrow \bullet}$  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  $\mathcal{C} \rightarrow \mathbf{Poly}$ , and that  $(\mathcal{C}, \mathcal{D})$ -bicomodules correspond to parametric right adjoint functors (prafunctors)  $\mathbf{Set}^{\mathcal{D}} \rightarrow \mathbf{Set}^{\mathcal{C}}$ . 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.

## 7.5 Exercise solutions

*Solution to Exercise 7.5.*

With  $q := y^2 + 3y^1$ , 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 descendants.

*Solution to Exercise 7.11.*

1. To select a 1-tree, we must select a position from  $1 \dots$  and then we are done, because there are no directions. So there is a unique 1-tree given by a root and no edges, implying that  $\text{tree}_1 \cong 1$ .
2. To select a 2-tree, we must select a position from 2. Then we are done, because there are no directions. So every 2-tree is a root with no edges, and the root is labeled with one of the elements of 2. Hence  $\text{tree}_2 \cong 2$ .
3. To select a  $y$ -tree, we must select a position from 1; then for the unique direction at that position, we must select a position from 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  $\text{tree}_y \cong 1$ .
4. To select a  $y^2$ -tree, we must select a position from 1; then for each direction at that position, we must select a position from 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  $\text{tree}_{y^2} \cong 1$ .
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  $f: \mathbb{N} \rightarrow 2$ , where  $f(n) \in 2$  is the label of the height- $n$  vertex of  $T$ . Hence  $\text{tree}_{2y} \cong 2^{\mathbb{N}}$ .
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  $\text{tree}_{y+1} \cong \mathbb{N} \cup \{\infty\}$  (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  $\sum_{n \in \mathbb{N}} A^n = \text{List}(A)$ . Then specifying a  $By^A$ -tree amounts to assigning a label from  $B$  to every vertex via a function  $\text{List}(A) \rightarrow B$ . Hence  $\text{tree}_{By^A} \cong B^{\text{List}(A)}$ .

*Solution to Exercise 7.12.*

1. Yes: by Exercise 7.11 #11 in the case of  $B = 1$ , or by analogy with Exercise 7.11 #7 and #8, we have that  $\text{tree}_{y^n}$  is the set of  $n$ -ary trees.
2. Yes: by Exercise 7.11 #11, we have that  $\text{tree}_{Ly^n}$  is the set of  $L$ -labeled  $n$ -ary trees.

*Solution to Exercise 7.14.*

We defined  $\text{tree}_p$  in Definition 7.2 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

$$1 \leftarrow p(1) \leftarrow p(p(1)) \leftarrow \dots$$

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  $i_2: p[i_1] \rightarrow p(1)$  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 7.16.*

1. To give a function  $t: \text{tree}_p \rightarrow p(\text{tree}_p)$  we need to choose, for each tree  $T \in \text{tree}_p$ , an element  $i \in p(1)$  and a function  $T.-: p[i] \rightarrow \text{tree}_p$ . 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  $f: S \rightarrow p \triangleleft S$ , we need to show that there is a unique function  $u: S \rightarrow \text{tree}_p$  that commutes with  $f$  and  $t$ , i.e. with  $t \circ u = p(f)$ .  
For each  $n$ , we have a function  $f_n: S \rightarrow p^{\triangleleft n} \triangleleft S$  by induction: for  $n = 0$  use the identity function  $S \rightarrow S$  and given  $S \rightarrow p^{\triangleleft n} \triangleleft S$  we compose with  $p^{\triangleleft n} \triangleleft f$  to get  $S \rightarrow p^{\triangleleft n} \triangleleft p \triangleleft S$ . Let  $f'_n: S \rightarrow p^{\triangleleft n}(1)$  be given by  $f'_n := f_n \circ (p^{\triangleleft n} \triangleleft !)$ . We know by Exercise 7.14 that  $\text{tree}_p$  is the limit of  $p^{\triangleleft n}(1)$ , so we get a unique map  $(f'_n)_{n \in \mathbb{N}}: S \rightarrow \text{tree}_p$ . 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  $\text{tree}_p \rightarrow p \triangleleft \text{tree}_p$  from the first part is invertible. Indeed, given  $i \in p(1)$  and  $T': p[i] \rightarrow \text{tree}_p$ , we construct an element of  $\text{tree}_p$  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 7.17.*

For the given values of  $p$ , we have already characterized the position-set  $t_p(1) \cong \text{tree}_p$  of  $t$  in Exercise 7.11, so it remains to characterize the directions at each position, i.e. the vertices of each  $p$ -tree.

1. We saw that  $\text{tree}_1 \cong 1$  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  $t_1 \cong 1y^1 \cong y$ .
2. We saw that  $\text{tree}_2 \cong 2$  and that each 2-tree has a single vertex, its root. Equivalently, its only rooted path is the empty path. So  $t_2 \cong 2y^1 \cong 2y$ .
3. We saw that  $\text{tree}_y \cong 1$  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  $t_y \cong y^{\mathbb{N}}$ .
4. We saw that  $\text{tree}_{y^2} \cong 1$  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  $\sum_{n \in \mathbb{N}} 2^n \cong \text{List}(2)$ . Equivalently, the rooted paths of an infinite binary tree are just finite binary sequences, which comprise the set  $\text{List}(2)$ . Hence  $t_{y^2} \cong y^{\text{List}(2)}$ .
5. We saw that  $\text{tree}_{2y} \cong 2^{\mathbb{N}}$ , and that every  $2y$ -tree is an infinite ray, whose vertices (or rooted paths) are in bijection with  $\mathbb{N}$ . Hence  $t_{2y} \cong 2^{\mathbb{N}}y^{\mathbb{N}}$ .
6. We saw that  $\text{tree}_{y+1} \cong \mathbb{N} \cup \{\infty\}$ . 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  $\infty$  is an infinite ray. Thus the direction-set of  $t_{y+1}$  at  $n \in \mathbb{N}$  is  $n+1$ , while its direction-set at  $\infty$  is  $\mathbb{N}$ . Hence  $t_{y+1} \cong \{\infty\}y^{\mathbb{N}} + \sum_{n \in \mathbb{N}} y^{n+1}$ .
7. We saw that  $\text{tree}_{By^A} \cong B^{\text{List}(A)}$ , 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  $\sum_{n \in \mathbb{N}} A^n = \text{List}(A)$ . Equivalently, the rooted paths of the  $By^A$ -tree are just finite sequences in  $A$ , which comprise the set  $\text{List}(A)$ . Hence  $t_{By^A} \cong B^{\text{List}(A)}y^{\text{List}(A)}$ .

*Solution to Exercise 7.36.*

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  $t_B(1) = \text{tree}_B$  is in bijection with  $B$  itself, yielding  $t_B \cong By$ . (We could have also applied Exercise 7.17 #7 in the case of  $A = 0$ , yielding  $t_{By^0} \cong B^{\text{List}(0)}y^{\text{List}(0)} \cong By$ , as  $\text{List}(0) \cong 1 + 0^1 + 0^2 + \dots \cong 1$ —the empty sequence is the only sequence with elements in 0.)
2. We characterize the  $B$ -tree category  $\mathcal{T}_B$  as follows. We saw in Exercise 6.27 and Exercise 6.36 that (up to isomorphism) there is a unique comonoid structure on  $By$ , corresponding to the discrete category on  $B$ . So  $\mathcal{T}_B$  must be isomorphic to the discrete category on  $B$ . In the language of  $B$ -trees, the objects of  $\mathcal{T}_B$  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 7.40.*

We characterize  $\mathcal{T}_{By^A}$ , following Example 7.39.

- The objects are  $By^A$ -trees. By Exercise 7.17 #7, or Exercise 7.12 #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  $B^{\text{List}(A)}$ .
- 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  $\text{List}(A)$ . 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 7.41.

We characterize  $\mathcal{T}_{y+1}$  as follows.

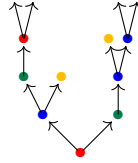
- The objects are  $(y+1)$ -trees. By Exercise 7.17 #6, a  $(y+1)$ -tree is either a single length- $n$  path for some  $n \in \mathbb{N}$ , which we denote by  $[n]$ , or a ray, which we denote by  $[\infty]$ . So the set of objects is  $\text{tree}_{y+1} = \{[n] \mid n \in \mathbb{N} \cup \{\infty\}\} \cong \mathbb{N} \cup \{\infty\}$ .
- 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  $\{0, \dots, n\}$ .  
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 7.42.

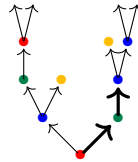
1. An object in  $\mathcal{T}_p$  is a stream of letters  $a, b, \dots, z$  and spaces  $\sqcup$ , that may go on forever or may end with a gigantic period, “•”. So an example is the infinite stream  $aaaaa \dots$ . Another example is  $hello \sqcup world \bullet$ .
2. The object  $aaaaa \dots$  has  $\mathbb{N}$  as its set of emanating morphisms. The object  $hello \sqcup world \bullet$  has  $\{0, \dots, 11\}$  as its set of emanating morphisms.
3. The codomain of any morphism out of  $aaaaa \dots$  is again  $aaaaa \dots$ . The codomain of any morphism  $0 \leq i \leq 11$  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 7.43.

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:



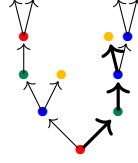
4. The codomain of  $f$  is the tree rooted at the target of the path, namely



5. Here's a morphism emanating from the codomain of  $f$ :



6. The composite morphism on the original tree is:



*Solution to Exercise 7.44.*

Take  $p := 2y$ , and consider the object  $x \in \mathcal{T}_p(1)$  given by the stream

$$x := (2\ 12\ 112\ 1112\ 11112\ 111112\ \dots)$$

(with spaces only for readability); note that every morphism emanating from  $x$  has a different codomain.

We need to give  $\varphi_i^\#(q)$  for every  $i \in \mathcal{T}_p(1)$  and  $q \geq 0$ . Define

$$\varphi_i^\#(q) := \begin{cases} i & \text{if } i \neq x \text{ or } q = 0 \\ x' & \text{if } i = x \text{ and } q > 0 \end{cases}$$

where  $x' := (12\ 112\ 1112\ 11112\ 111112\ \dots)$ . There are three cofunctor conditions to check, namely identity, codomains, and composition. The codomain condition is vacuous since  $y^Q$  has one object, and the identity condition holds by construction, because we always have  $\varphi_i^\#(0) = i$ . Now take  $q_1, q_2 \in Q$ ; we need to check that

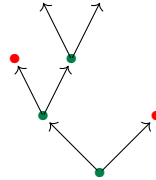
$$\varphi_{\text{cod } \varphi_i^\#(q_1)}^\#(q_2) \stackrel{?}{=} \varphi_i^\#(q_1 + q_2)$$

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 7.53.*

Recall that  $p := \{\bullet\}y^2 + \{\bullet\}$  and  $q := \{\bullet, \bullet\}y + \{\bullet, \bullet\}$ .

1. Have  $\varphi: p \rightarrow q$  send forward  $\bullet \mapsto \bullet$ , with the unique element of  $q[\bullet]$  sent back to the left branch of  $p[\bullet]$ , and send forward  $\bullet \mapsto \bullet$ .
2. Here is a sample  $T \in \text{tree}_p$ .



3. Here is  $\mathcal{T}_\varphi(T)$ :



*Solution to Exercise 7.54.*

1. By the adjunction from Theorem 7.45, cofunctors  $\mathcal{T}_p \rightarrow \mathcal{T}_{p^{\text{an}}}$  are in bijection with polynomial maps  $\mathfrak{t}_p \rightarrow p^{\text{an}}$ , where  $\mathfrak{t}_p = U \mathcal{T}_p$  is the carrier. From Proposition 6.20 we have a map  $\delta^{(n)}: \mathcal{T}_p \rightarrow \mathcal{T}_{p^{\text{an}}}$ , and we also have the counit of the cofree adjunction which we'll temporarily call  $r: \mathcal{T}_p \rightarrow p$ . Then the desired map is  $(\delta^{(n)} \circ r^{\text{an}}): \mathfrak{t}_p \rightarrow p^{\text{an}}$ .

- When  $n = 1$  the induced cofunctor is an isomorphism, but for  $n \geq 2$  it is not. However, we should start by saying that the function on objects  $\text{tree}_p \rightarrow \text{tree}_{p^{*n}}$  is a bijection. It sends a  $p$ -tree to the  $p^{*n}$  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  $\text{tree}_{p^{*n}}$  would correspond to a rooted path on  $\text{tree}_p$  whose length is a multiple of  $n$ . When  $n = 2$  and  $p = y + 1$ , some rooted paths in  $\text{tree}_p$  have odd lengths.

*Solution to Exercise 7.62.*

A category  $Sy$  with linear carrier is a discrete category on  $S$ , so we need to show that a cofunctor between the discrete category on  $S$  and the discrete category on  $T$  is the same thing as a function  $S \rightarrow T$ . But this is clear: a cofunctor  $Sy \rightarrow Ty$  is a function  $S \rightarrow T$  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 7.65.*

- Given an object  $i \in c'(1) = c(1)$ , the set of emanating morphisms is  $c'[i] := \mathfrak{d}[fi]$ , and they compose according to the structure of  $\mathfrak{d}$ . Since morphisms in  $\mathfrak{d}$  compose associatively and unitaly, so do morphisms in  $c'$ .
- Given  $i \in c(1)$  and  $m \in c'[i] = \mathfrak{d}[fi]$ , we have  $g(\text{cod}(g_i^\#(m))) = \text{cod}(g_i^\#(m)) = \text{cod}(f_i^\#(m)) = \text{cod}(m)$  by definition.
- For any  $i \in c'(1)$ , the lens  $g_i^\#$  preserves compositions of morphisms in  $c'[i] = \mathfrak{d}[fi]$  because it agrees with  $f_i^\#$ , which preserves compositions.
- Given  $i \in c'(1)$  and  $m \in \mathfrak{d}[hi]$ , we have  $h(\text{cod}(h_i^\#(m))) = f(\text{cod}(f_i^\#(m))) = m$ .
- The map  $h_i^\#$  was chosen to be the identity for each  $i$ , so it certainly preserves compositions.

*Solution to Exercise 7.68.*

The category corresponding to  $Sy^S$  is the contractible groupoid  $K_S$  on  $S$ . A vertical cofunctor  $K_S \rightarrow \mathcal{C}$  includes a bijection on objects, so we can assume that  $\text{Ob}(\mathcal{C}) = S$ . But the rest of the cofunctor assigns to each map  $s_1 \rightarrow s_2$  in  $\mathcal{C}$  some choice of morphism  $s_1 \rightarrow s_2$  in  $K_S$ , and there is exactly one. Thus the cofunctor includes no additional data.

*Solution to Exercise 7.69.*

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  $c := y^{\{\text{id}_1, s, t\}} + y^{\{\text{id}_2\}}$  and that of  $\mathcal{D}$  is  $\mathfrak{d} := y^{\{\text{id}_1, f\}} + y^{\{\text{id}_2\}}$ .

- The boo functor  $F: \mathcal{C} \rightarrow \mathcal{D}$  can be identified with a vertical cofunctor  $\mathfrak{d} \rightarrow c$  that sends back  $\text{id}_1 \mapsto \text{id}_1$  and  $s \mapsto f$  and  $t \mapsto f$  and  $\text{id}_2 \mapsto \text{id}_2$ .
- For  $G_1: \mathcal{D} \rightarrow \mathcal{C}$ , which we take to send  $f \mapsto s$ , the corresponding vertical cofunctor  $c \rightarrow \mathfrak{d}$  sends back  $\text{id}_1 \mapsto \text{id}_1$  and  $f \mapsto s$  and  $\text{id}_2 \mapsto \text{id}_2$ .

*Solution to Exercise 7.74.*

- We actually already showed that  $0$  has a unique comonoid structure, corresponding to the empty category (which we will also denote by  $0$ ), in Exercise 6.39, for the case of  $S := 0$ .
- We show that  $0$  has the universal property of the initial object in  $\mathbf{Cat}^\#$  as follows. For any category  $\mathcal{D}$ , there is a unique cofunctor  $0 \rightarrow \mathcal{D}$ : 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 7.72 that  $\mathbf{Cat}^\#$  has an initial object, and that the forgetful functor  $\mathbf{Cat}^\# \rightarrow \mathbf{Poly}$  sends it to the initial object in  $\mathbf{Poly}$ , which is  $0$  (by Proposition 2.51 and the following discussion). So the initial object in  $\mathbf{Cat}^\#$  must be the unique comonoid carried by  $0$ .

*Solution to Exercise 7.75.*

The coproduct of  $c$  with itself is again a comonoid (as a category it is the disjoint union of  $c$  with itself) and it has the right carrier polynomial, since the carrier functor  $U: \mathbf{Cat}^\# \rightarrow \mathbf{Poly}$  is a left adjoint.



Solution to Exercise 7.78.

1. The highbrow way to check that  $\mathfrak{c} \otimes \mathfrak{d}$  forms a comonoid is to say that

$$\otimes: (\mathbf{Poly}, y, \triangleleft) \times (\mathbf{Poly}, y, \triangleleft) \longrightarrow (\mathbf{Poly}, y, \triangleleft)$$

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 **Cat**. This is a category, hence a comonoid in **Poly**.

2. Because  $\otimes$  is symmetric, it suffices to show that for any cofunctor  $\varphi: \mathcal{C} \rightarrow \mathcal{C}'$ , there is an induced cofunctor  $\varphi \otimes \mathcal{D}: \mathcal{C} \otimes \mathcal{D} \rightarrow \mathcal{C}' \otimes \mathcal{D}$ .

Solution to Exercise 7.80.

1. The carrier of  $\mathcal{T}_y$  is  $\mathfrak{t}_y := y^{\mathbb{N}}$ .
2. The map  $y \rightarrow \mathfrak{t}_y$  is the unique map.
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 7.83.

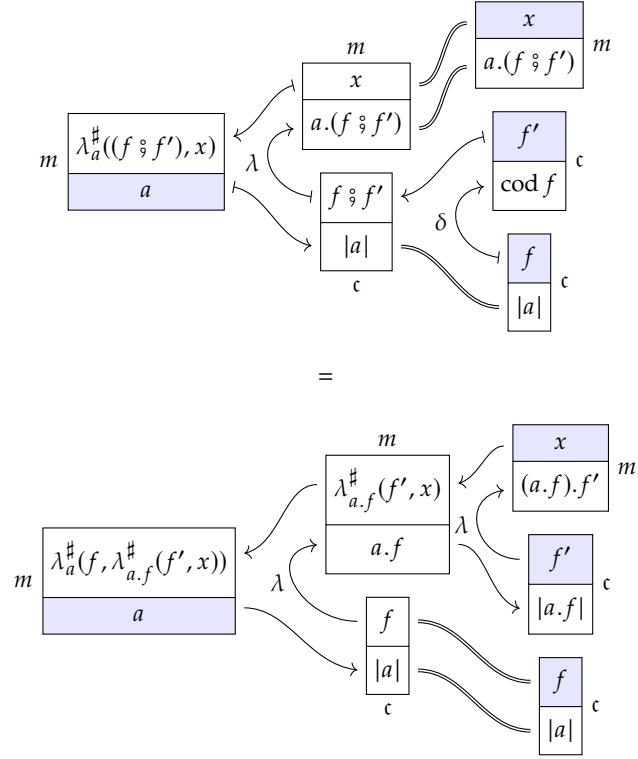
In Definition 7.81, if the carrier of the left  $\mathcal{C}$ -comodule is always chosen to be a constant polynomial, i.e. a set, then we recover Definition 6.95. Hence constant left  $\mathcal{C}$ -comodules are precisely left  $\mathcal{C}$ -coalgebras; their notions of morphisms coincide as well, so the categories they form are isomorphic.

Solution to Exercise 7.86.

1. Here is the first equation in terms of polyboxes:

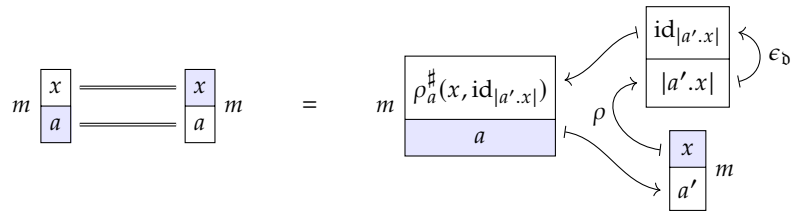
The equation says that for any  $a \in m(1)$  we have  $a = a.\text{id}_{|a|} := \lambda(a, \text{id}_{|a|})$  and that for any  $x \in m[a]$  we have  $x = x' := \lambda_a^\#(\text{id}_{|a|}, x)$ .

Here is the second equation in terms of polyboxes:



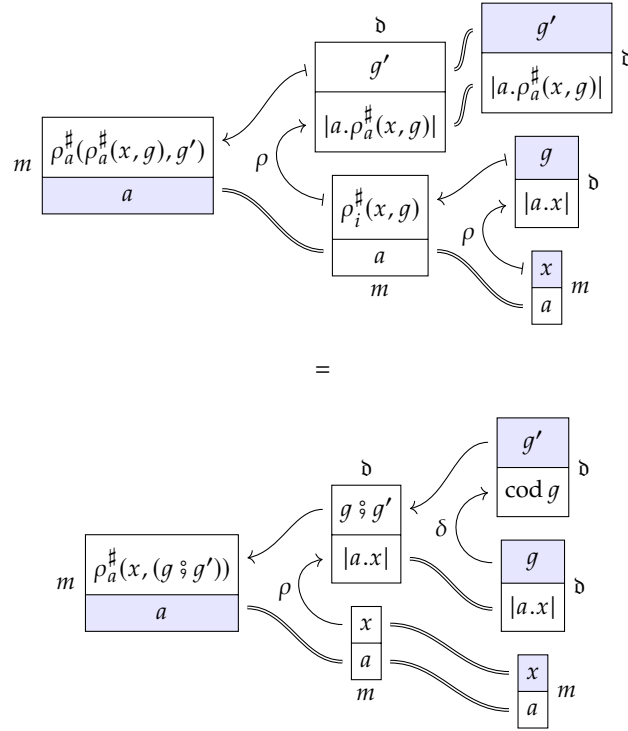
These say that  $\text{cod}(f) = |a.f|$ , that  $a.(f ; f') = (a.f).f'$ , and that  $\lambda_a^{\#}((f ; f'), x) = \lambda_a^{\#}(f, \lambda_{a.f}^{\#}(f', x))$ .

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  $x = \rho_a^{\#}(x, \text{id}_{|a'.x|})$ .

Here is the second equation in terms of polyboxes:



These equations say that  $\text{cod}(g) = |a.\rho_a^\#(x, g)|$  and that  $\rho_a^\#(\rho_a^\#(x, g), g') = \rho_a^\#(x, (g \circ g'))$ .

*Solution to Exercise 7.87.*

The comonoid structure on  $y$  has  $\epsilon: y \rightarrow y$  and  $\delta: y \rightarrow y \triangleleft y = y$  the unique maps.

1. We first show that for any polynomial  $m$ , there is a unique  $y$ -comodule structure on  $m$ . By (7.82), a  $y$ -comodule structure is a map  $\lambda: m \rightarrow y \triangleleft m = m$  such that  $\lambda \circ (! \triangleleft m) = \text{id}_m$ , but  $! \triangleleft m = \text{id}_m$ , so  $\lambda = \text{id}_m$  is forced. A morphism  $(m, \text{id}) \rightarrow (n, \text{id}_n)$  of  $y$ -comodules is also easily seen to be just a polynomial map  $m \rightarrow n$ .
2. Similar.

*Solution to Exercise 7.99.*

1. By Exercise 7.87, it suffices to check that the diagram (7.97) commutes vacuously when  $\mathfrak{d} = y$ .
2. By Exercise 7.87, it suffices to check that the diagram (7.97) commutes vacuously when  $\mathfrak{c} = y$ .
3. This follows from Exercise 7.87.
4. This follows from Exercise 7.87.

*Solution to Exercise 7.107.*

For each  $v \in V$ , let  $A_v := \text{src}^{-1}(v)$ , the set of arrows emanating from  $v$ .

1. We need a map  $\sum_{v \in V} y^{A_v} \rightarrow V \sum_{v \in V} y^{A_v}$ , and we use the obvious “diagonal” map, which on directions sends  $v \mapsto (v, v)$  and on directions is the identity.
2. We need a map  $\sum_{v \in V} y^{A_v} \rightarrow \sum_{v \in V} \prod_{a \in A_v} \sum_{v'' \in V} y$ , 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  $A_v \rightarrow V$ , 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  $2V \rightarrow V \times 2V$  to be  $(i, v) \mapsto (v, i, v)$  for any  $i \in 2$  and  $v \in V$ .

*Solution to Exercise 7.110.*

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  $V \rightarrow T$  between them chooses one color for each vertex, which we’re calling the “starting color”.

---

## 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  $\mathbf{Set} \rightarrow \mathbf{Set}$ , e.g. ones that aren't polynomial.
2. What can you say about the internal logic for the topos  $[\mathcal{T}_p, \mathbf{Set}]$  of dynamical systems with interface  $p$ , in terms of  $p$ ?
3. How does the logic of the topos  $\mathcal{T}_p$  help us talk about issues that might be useful in studying dynamical systems?
4. Morphisms  $p \rightarrow q$  in  $\mathbf{Poly}$  give rise to left adjoints  $\mathcal{T}_p \rightarrow \mathcal{T}_q$  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:  $\mathbf{Poly}$ ,  $\mathbf{Cat}^\#$ , and  $\mathbf{Mod}$ . Find examples of these comonoids, and perhaps characterize them or create a theory of them.
6. The category  $\mathbf{Poly}$  has pullbacks, so one can consider the bicategory of spans in  $\mathbf{Poly}$ . Is there a functor from that to  $\mathbf{Mod}$  that sends  $p \mapsto \mathcal{T}_p$ ?
7. Databases are static things, whereas dynamical systems are dynamic; yet we see them both in terms of  $\mathbf{Poly}$ . 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  $\mathbf{Set} \rightarrow \mathbf{Set}$ , what happens if we replace sets with homotopy types: how much goes through? Is anything improved?
10. Are there any functors  $\mathbf{Set} \rightarrow \mathbf{Set}$  that aren't polynomial, but which admit a

comonoid structure with respect to composition  $(y, \triangleleft)$ ?

11. Characterize the monads in  $\mathbf{poly}$ . 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  $\mathbf{Cat}^\#$  combinatorially.
13. Since the forgetful functor  $U: \mathbf{Cat}^\# \rightarrow \mathbf{Poly}$  is faithful, it reflects monomorphisms: if  $f: \mathcal{C} \rightarrow \mathcal{D}$  is a cofunctor whose underlying map on carriers is monic, then it is monic. Are all monomorphisms in  $\mathbf{Cat}^\#$  of this form?
14. Are there polynomials  $p$  such that one use something like Gödel numbers to encode logical propositions from the topos  $[\mathbf{tree}_p, \mathbf{Set}]$  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. 48).
- [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. 34).
- [Abb03] Michael Gordon Abbott. “Categories of Containers”. PhD thesis. University of Leicester, Aug. 2003 (cit. on p. 48).
- [Agu97] Marcelo Aguiar. *Internal categories and quantum groups*. Cornell University, 1997 (cit. on pp. 203, 218).
- [AU16] Danel Ahman and Tarmo Uustalu. “Directed Containers as Categories”. In: *EPTCS 207, 2016*, pp. 89–98 (2016). eprint: [arXiv:1604.01187](https://arxiv.org/abs/1604.01187) (cit. on p. 218).
- [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. 89).
- [Cap22] Matteo Capucci. *Diegetic representation of feedback in open games*. 2022. URL: <https://arxiv.org/abs/2206.12338> (cit. on p. 112).
- [Con12] John Horton Conway. *Regular algebra and finite machines*. Courier Corporation, 2012 (cit. on p. 89).
- [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. 279).
- [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. 48, 151, 157).

- [Hed18] Jules Hedges. *Limits of bimorphic lenses*. 2018. eprint: [arXiv:1808.05545](https://arxiv.org/abs/1808.05545) (cit. on p. 36).
- [Jac17] Bart Jacobs. *Introduction to Coalgebra*. Vol. 59. Cambridge University Press, 2017 (cit. on pp. 48, 149).
- [Kel74] G Max Kelly. “Doctrinal adjunction”. In: *Category seminar*. Springer, 1974, pp. 257–280 (cit. on p. 267).
- [Lia22] Amélia Liao. *Cat.instances.poly*. Aug. 2022. URL: <https://11lab.dev/Cat.Instances.Poly.html> (cit. on p. 11).
- [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. 34).
- [MM92] Saunders MacLane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Springer, 1992 (cit. on p. 25).
- [Mye22] David Jaz Myers. *Categorical Systems Theory*. 2022 (cit. on p. vii).
- [nLa18] Contributors To nLab. *Created limit — nLab*. 2018. URL: <https://ncatlab.org/nlab/show/created+limit> (cit. on p. 265).
- [nLa19] Contributors To nLab. *Connected limit — nLab*. 2019. URL: <https://ncatlab.org/nlab/show/connected+limit> (cit. on p. 153).
- [nLa22] nLab authors. *lens (in computer science)*. <http://ncatlab.org/nlab/show/lens%20in%20computer%20science%29>. Revision 26. Aug. 2022 (cit. on p. 218).
- [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. 265).
- [Por19] Hans-E Porst. “Colimits of monoids”. In: *Theory and Applications of Categories* 34.17 (2019), pp. 456–467 (cit. on pp. 265, 266).
- [Sha21] Brandon Shapiro. *Familial Monads as Higher Category Theories*. 2021. URL: <https://arxiv.org/abs/2111.14796> (cit. on p. 279).
- [Shu08] Michael Shulman. “Framed bicategories and monoidal fibrations”. In: *Theory and Applications of Categories* 20 (2008), Paper No. 18, 650–738 (cit. on pp. 108, 111, 112).
- [Spi22] David I. Spivak. *Polynomial functors and Shannon entropy*. 2022. arXiv: [2201.12878](https://arxiv.org/abs/2201.12878) [math.CT] (cit. on p. 112).
- [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. 89).
- [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. 279).