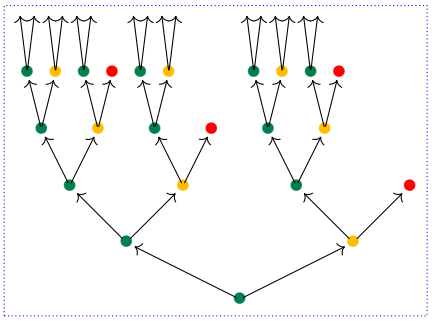
DRAFT

Categorical Systems Theory



David Jaz Myers David I. Spivak

(Last updated: October 2, 2020)

This page intentionally left blank.

To André Joyal and Bill Lawvere

Preface

Organization is form: it's the form something takes to carry out its mission. Said another way, a mission, a coherent set of values and techniques, is something around which an organization can be formed. Organizations channel something into the world.

The world's most beautiful organizational marvels—from cats and dogs to people and effective altruistic organizations—can sometimes be seen to achieve their own life purposes, to be like a dog playing with other dogs on a beach. And then they are a pleasure to behold, a display of the abundance that can be found when things just work.

What is that abundance and play of forms? How do we talk about it in a way that borders between poetic and deeply meaningful on the one hand, and plain and open for investigation on the other? We long to participate together in a conversation about our love: the play that can happen when one is truly at home somewhere. Do you see the dogs playing on the beach?!

We believe a deeper conversation is possible, but it may require the development of appropriate language, good distinctions to have for a sustained consideration of the play of life. How would you create language by which we could see the game of catch being played by two friends? How would you talk about a clock and a counter that could help us track changes? How could you plug together simpler things and fashion a program for the way they'll interact?

Suppose some people want to create the best conceptual tools for this play. Where would they start? Our answer is to start where we are: two mathematicians who like to play. We invite you to nothing more than a play of forms that we ourselves find both fun and important to the pursuit of values.

For better or for worse, we start from an advanced level: one of us has a PhD, and the other will likely have one soon. We have studied a language form called category theory, to which we are deeply indebted. Our thinking about reality is steeped in this language form, because we find it so effective in helping us talk about things we care about.

We can never get *at* what we deeply are within a finite language, but we can talk *about* it, dance around it and point like we're touched by something unseen. Here we

attempt to invite you in and sketch a vehicle for your own journey, at least for a phase along its way.

Acknowledgments

Thanks go to Joachim Kock, Sophie Libkind, Nelson Niu, Richard Garner, Todd Trimble.

Contents

Chapter 1

Wiring together dynamical systems

1.1 Introduction

Here's a basic fact of life: *things change*. And how things change most often depends on how they currently are. This is the basic idea underlying all the various notions of *dynamical system* that we will see in this book.

Informal Definition 1.1. A *dynamical system* consists of:

- a notion of how things can be, called the *states*, and
- a notion of how things will change given how they are, called the *dynamics*.

The dynamics of a system might also depend on some free *parameters* or *inputs* that are imported from the environment, and we will often be interested in some particular *variables* of the state that are *exposed* or *output* to the environment.

You and I are big, complicated dynamical systems. Our bodies and minds are in some particular configuration, and over time this configuration changes. We can sense things — seeing, touching, tasting — and what we sense affects how our bodies and mind changes. Seeing a scary snake can make me recoil and feel fear, but seeing a cute snake plushie can make me go over and start to pet it. Some parts of me are also put back into the environment, like the expression on my face. But not all of me is exposed in that way — some things just go on in my head.

This is the basic model of a dynamical system we will be working with in this book. But to make the above informal definition precise, we need to answer a number of questions:

- What should a state be, really? Do we just have an abstract set of states, or could there be a continuum of states? Maybe there are some other structures that states can enter into which have to be respected by the dynamics, but aren't determined by them?
- What does it mean to change? Do we want to know precisely which state will be next if we know how things are? Or, maybe we will only have a guess at which

state will come next? Or, maybe we'll just say how a state is tending to change, but not where it will end up?

- Do we always take in the same sort of parameters, or does it depend on how our system is placed in its environment? Should the dynamics vary continuously (or linearly, or some other way) in the choice of parameters?

Different people have decided on different answers to these questions for different purposes. Here are some of the most widespread ways to answer those questions:

1. We'll assume the states form a discrete set, and that if we know the current state and our parameters, we know exactly what the next state will be. Such a system generally called a *Moore machine*.
2. We'll assume the states form a continuum, but that we only know how a state is tending to change, not what the "next" state will be. Such a system is generally called a *system of differential equations* — the differential equations tells us the way the derivatives of the state variables, the way they are tending.
3. We'll assume the states form a discrete set, but that we only have a guess at which state will follow from the current state. Such a system is generally called a *Markov process*, or a *Markov decision process*.

We will call a way of answering these questions the *doctrine* of dynamical systems we are working in.

Informal Definition 1.2. A *doctrine* of dynamical systems is a particular way to answer the following questions about what it means to be a dynamical system:

- What does it mean to be a state?
- How should the output vary with the state — discretely, continuously, linearly?
- Can the kinds of input a system takes in depend on what it's putting out, and how do they depend on it?
- What sorts of changes are possible in a given state?
- How should the way the state changes vary with the input?

Moore machines, differential equations, and Markov decision processes are each dynamical systems understood in a different doctrine.

1. A Moore machine is a dynamical system in a *discrete and deterministic* doctrine.
2. A system of differential equations is a dynamical system in a *differential* doctrine.
3. A Markov decision process is a dynamical system in a *stochastic* doctrine.

In most cases, mathematicians have assumed that that the kinds of parameters our systems take in never change — that our system will always interface with its environment in the same way. However, this assumption is quite restrictive; after all, I change the way I interface with my environment all the time. Every time I turn and face a new direction, I open myself up to new inputs. There are variations on all of the above doctrines which allow for the kinds of input to depend on what the system is putting out, but in this book we will take a deep dive into the discrete and deterministic variant.

4. A *dependent system* is a dynamical system in a (discrete, deterministic, and) *dependent* doctrine.

The dynamical systems we will see in this book are *open* in the sense that they take in inputs from their environment and expose outputs back to their environment. Because of this, our systems can interact with eachother. One system can take what the other system outputs as part of its input, and the other can take what the first outputs as part of its input. For example, when we have a conversation, I take what I hear from you and use it to change how I feel, and from those feelings I generate some speech which I output to the world. You then take what I've said and do the same thing.

We call this way of putting together dynamical systems to make more complex systems *composition*.

Informal Definition 1.3. *Composition* is the process by which some things are brought together to form bigger things.

Functions can be composed by $g \circ f(x) = g(f(x))$, and dynamical systems can be composed by plugging in the variables of the states of some into the parameters of others.

This book is all about composing dynamical systems.
Because of this, we will use the abstract language of composition: *category theory*.

Informal Definition 1.4. *Category theory* is the abstract study of composition.

1.1.1 Category Theory

We'll be using the language of category theory quite freely in this book, and so we'll expect you to know the basics. These are the notions we will expect you to be familiar with:

- What a category is.
- What an isomorphism is.
- What a functor is.
- What a natural transformation is.
- What a terminal and an initial object are.
- What a product and a coproduct are.
- What a monad is, and it will help if you also know what a comomad is.
- What a monoidal category is.

Good introductions to category theory abound. One place to start is *An invitation to applied category theory* [**fong2019seven**].

We will be using cartesian categories quite a bit in the first few chapters.

Definition 1.5. A category $\mathcal{C}$ is *cartesian* if every two objects A and B in $\mathcal{C}$ have a product $A \times B$, and $\mathcal{C}$ has a terminal object 1. Equivalently, $\mathcal{C}$ is cartesian if for any finite set I and I-indexed family $A_{(-)} : I \to \mathcal{C}$ of objects, there is a product $\prod_{i \in I} A_i$ in $\mathcal{C}$.

A functor $F : \mathcal{C} \to \mathcal{D}$ between cartesian categories is said to be *cartesian* if it preserves products and terminal objects, i.e. the map $(F\pi_A, F\pi_B) : F(A \times B) \to FA \times FB$ is an isomorphism for all A and B, and the terminal morphism $F1 \to 1$ is an isomorphism.

We will also use some more advanced category theory, like indexed categories, double categories, and toposes. However, you don't need to know them up front; we will introduce these concepts as we use them.

While we're at it, here's some notation we'll use repeatedly throughout the book. The nth ordinal is denoted n. It is defined to be the set

$$\mathsf{n} := \{1, 2, \ldots, n\}.$$

So $\mathsf{0}$ is the empty set, $\mathsf{1}$ is a one-element set, etc.

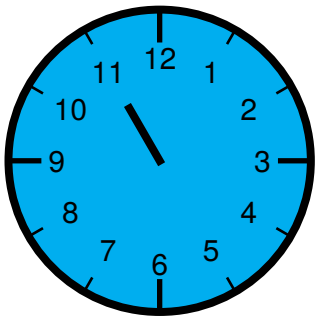
1.2 Deterministic and differential doctrines

In this chapter, we will see how to wire together dynamical systems of all different sorts. First, however, we start with two exemplary doctrines:

1. First, systems which we will call *(discrete-time) deterministic systems*, which specify exactly which state the system will transition into given its current state and input parameters.
2. Second, systems which we will call *differential systems*, which do not specify a "next state" but rather specify exactly how the state is tending to change in the moment, given the current state and input parameters.

1.2.1 Deterministic systems

A paradigmatic example of this sort of dynamical system is a clock.



Suppose that our clock has just an hour hand for now. Then we may collect all the way things can be for the clock into a set of hours:

$$\mathsf{Hour} := \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}.$$

This set Hour is the set of *states* of our clock system.

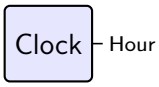
If we know what hour it is, we also know what hour is coming next. So, this system has the following dynamics:

$$\text{tick} : \text{Hour} \longrightarrow \text{Hour} \tag{1.6}$$
$$t \longmapsto \begin{cases} t+1 & \text{if } t < 12 \\ 1 & \text{if } t = 12 \end{cases}$$

Here's a sample of the dynamics of the clock. Say we started at the 10 o'clock state:

$$10 \overset{\text{tick}}{\longmapsto} 11 \overset{\text{tick}}{\longmapsto} 12 \overset{\text{tick}}{\longmapsto} 1 \overset{\text{tick}}{\longrightarrow} 2 \overset{\text{tick}}{\longmapsto} \cdots$$

Ok, it's not the most dynamic of systems, but we have to start somewhere. If we want to refer to the whole system at once, we can box it up and draw it like this:



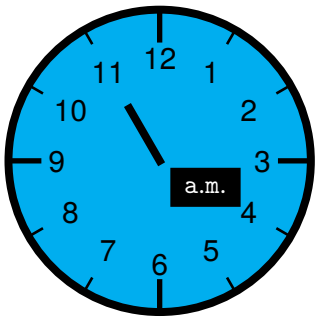
(1.7)

We imagine that the clock is going about its business inside the box, and that is shows the hour it is currently displaying on the outgoing wire. This outgoing wire constitutes the clock's exposed variable, but we'll explain that more later.

One issue with our clock is that it doesn't tell us whether it is morning or evening. Being morning or evening and going back and forth between them is another way that things might be and change, and hence we can see it as its own two-state dynamical system with states

$$\text{a.m./p.m.} = \{\texttt{a.m.}, \texttt{p.m.}\}.$$

However, rather than have this be an independent system, we want to consider it as a little addition to our clock system, one that reads a.m. or p.m.:



(1.8)

To connect the meridian to the clock means that the way the meridian changes should be based on the hour:

$$\text{next} : \text{a.m./p.m.} \times \text{Hour} \longrightarrow \text{a.m./p.m.} \tag{1.9}$$
$$(\texttt{a.m.}, t) \longmapsto \begin{cases} \texttt{p.m.} & \text{if } t = 11 \\ \texttt{a.m.} & \text{otherwise} \end{cases}$$

$$(\texttt{p.m.}, t) \mapsto \begin{cases} \texttt{a.m.} & \text{if } t = 11 \\ \texttt{p.m.} & \text{otherwise} \end{cases}$$

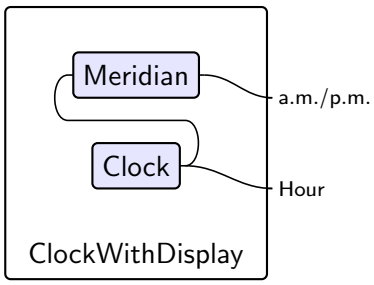
If it is a.m. and the clock reads 8, then it will still be a.m. at the next tick; but if it is a.m. and the clock reads 11, then the next tick will switch the meridian to p.m..

Again, the thing to note about the dynamics of the a.m./p.m. system is that they depend on what hour it is. The hour is imported as a *parameter* for the dynamics of the meridian system. We can draw the meridian system as a box like this:

(1.10)

We have the a.m./p.m. wire coming out, which carries the information of whether it is a.m. or p.m., just like the clock. But we also have a wire coming in, which carries the hour that we need as a parameter for our dynamics.

We can now express our whole clock (1.8) by wiring together our bare clock (1.7) and the a.m./p.m. system:



(1.11)

The resulting system has states

$$\mathsf{HoursWithDisplay} := \mathsf{Hour} \times \mathsf{a.m./p.m.}$$

each of which is a pair, e.g. (11, a.m.), consisting of an hour and a meridian reading. They update in a combined way, by using the hour shown on the clock face as the parameter we need for the Meridian system; this is expressed by having a wire from the output of Clock to the input of Meridian. In full, the dynamics looks like this:

$$\begin{aligned} \mathsf{tick'} : \mathsf{HoursWithDisplay} &\to \mathsf{HoursWithDisplay} \\ (t, m) &\mapsto (\mathrm{tick}(t), \mathrm{next}(t, m)) \end{aligned}$$

where tick and next are as in (1.6) and (1.9).

Exercise 1.12. Expand the definition of the combined system out in full, and check that it really does behave like the clock with a.m./p.m. display should. ◊

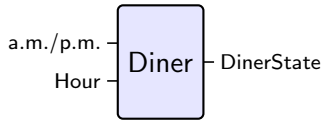
Now that we have a working clock, we can use it for systems that need to know the time. For example, consider a diner that opens at 7a.m. and closes at 10p.m.. The states of this diner are

$$\mathsf{DinerState} = \{\texttt{open}, \texttt{closed}\}.$$

The diner's dynamics are then

$$\begin{aligned} \text{dinerDynamics} : \mathsf{DinerState} \times \mathsf{HoursWithDisplay} &\to \mathsf{DinerState} \\ (\texttt{open}, (10, \texttt{p.m.})) &\mapsto \texttt{closed} \\ (\texttt{closed}, (7, \texttt{a.m.})) &\mapsto \texttt{open} \\ (s, (t, m)) &\mapsto s \quad \text{otherwise.} \end{aligned}$$

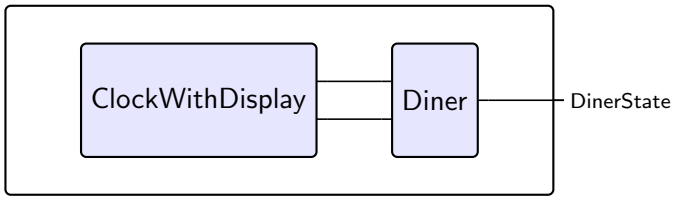
Again, we can represent the diner by this box:



(1.13)

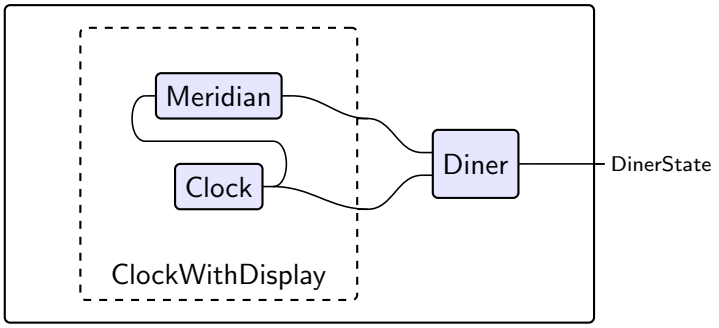
This time, we have two wires coming in, corresponding to the two parameters we need for the diner system: the hour and the meridien.

Assuming that the diner has a clock on its wall which it uses to decide whether to open or close, the full diner system would be given by wiring the clock with display into those input wires:



(1.14)

If we want to, we can peak into the clock with display and see that it is itself made out of a clock wired to a display:



(1.15)

These examples are simple, but it doesn't take much more to get to some truly amazing phenomena. Consider this system: we have an infinite tape with a read-head at some integer position. On this infinite tape, we will write the symbols a, b, c, or d, or we will leave it blank: _. Together, the state of the tape and the position of the read-head have states pairs (T, n) consisting of a function $\mathrm{T} : \mathbb{Z} \to \{a, b, c, d, \_\}$, telling us what symbol $\mathrm{T}(i)$ is found at position i of the tape, and a position n of the read-head:

$$\begin{aligned} \mathsf{Symbol} &= \{a, b, c, d, \_\} \\ \mathsf{Tape} &= \mathsf{Symbol}^{\mathbb{Z}} \end{aligned}$$

$$\mathsf{Head} = \mathbb{Z}$$

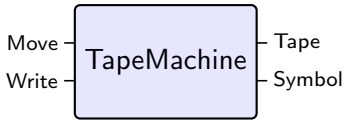
The parameters that this system needs in order to change are a move-command and a write-command. The move-command will be either move left or move right, encoded as -1 or 1 respectively, and the write command will be one of the symbols that can be written on the tape:

$$\mathsf{Move} = \{-1, 1\} \qquad \text{and} \qquad \mathsf{Write} = \{a, b, c, d, \_\}.$$

The way this system changes is by writing the write command to the tape at the current position, and then moving according to the move command. As a function, this is:

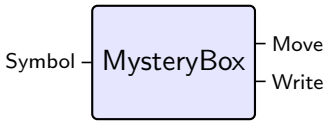
$$\begin{aligned} &\text{execute} : \mathsf{Head} \times \mathsf{Tape} \times \mathsf{Move} \times \mathsf{Write} \longrightarrow \mathsf{Head} \times \mathsf{Tape} \\ &\qquad (n, i \mapsto \mathrm{T}(i), d, s) \mapsto \left(n + d, i \mapsto \begin{cases} \mathrm{T}(i) & \text{if } i \neq n \\ s & \text{if } i = n \end{cases} \right). \end{aligned}$$

We can imagine that the system exposes the tape and the symbol under its read head. We can box this system up and draw it like so:



(1.16)

Now, we need one more simple ingredient to get our system going; a mysterious system of the form:



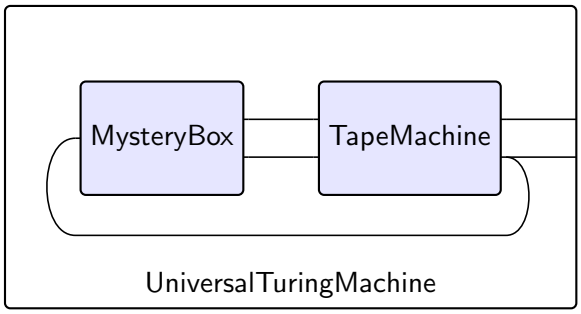
(1.17)

We can see that our mystery box will take in a symbol and put out a move command and a write command. The way our mystery box behaves is rather mysterious. It has six states $S \cong \mathsf{6}$, and its update rule is given by the following table, where the entry in the row i and the column s is written $(m, w) : s'$ to express the move command m, the write command w, and the next state s' that our mysterious system transitions to when input the symbol i in state s:

	1	2	3	4	5	6
a	(-1, b):1	(1, a):1	(-1, b):3	(1, b):2	(-1, b):6	(-1, b):4
b	(-1, a):1	(1, a):2	(-1, b):5	(1, a):4	(1, a):6	(1, a): 5
c	(1, d):2	(1, d):2	(-1, c):5	(1,d):4	(1, c):5	(1, a):1
d	(-1, c):1	(1, a):5	(-1, c):3	(1,d):5	(-1, b):3	end

(1.18)

Mysterious indeed. But when we wire the two together, magic happens!



(1.19)

This is a universal Turing machine, i.e. when we encode everything into this strange alphabet, it is capable of arbitrarily complex calculation!

Even simple systems can have very interesting behavior when plugged in to the right environment.

That's a lot of informal definitions, we are ready for something precise:

Definition 1.20. A *deterministic system* S, also written as

$$\begin{pmatrix}\mathsf{update}_\mathsf{S}\\ \mathsf{expose}_\mathsf{S}\end{pmatrix} : \begin{pmatrix}\mathsf{State}_\mathsf{S}\\ \mathsf{State}_\mathsf{S}\end{pmatrix} \leftrightarrows \begin{pmatrix}\mathsf{In}_\mathsf{S}\\ \mathsf{Out}_\mathsf{S}\end{pmatrix},$$

consists of:

- a set $\mathsf{State}_\mathsf{S}$ of *states*;
- a set Out_S of *values for exposed variables*, or *outputs* for short;
- a set In_S of *parameter values*, or *inputs* for short;
- a function $\mathsf{expose}_\mathsf{S} : \mathsf{State}_\mathsf{S} \to \mathsf{Out}_\mathsf{S}$, the *exposed variable of state* or *expose* function, which takes a state to the output it yields; and
- a function $\mathsf{update}_\mathsf{S} : \mathsf{State}_\mathsf{S} \times \mathsf{In}_\mathsf{S} \to \mathsf{State}_\mathsf{S}$, the *dynamics* or *update* function which takes a state and a parameter and gives the next state.

We refer to the pair $\left(\begin{smallmatrix}\mathsf{In}_\mathsf{S}\\ \mathsf{Out}_\mathsf{S}\end{smallmatrix}\right)$ of exposed variable and parameter values as the *interface* of the system.

We can interpret this definition in any cartesian category $\mathcal{C}$; here, we have have used the category **Set** of sets.

Remark 1.21. Deterministic systems are also known as *Moore machines* in the literature. If the output set is taken to be $\{\texttt{true}, \texttt{false}\}$, then they are known as *deterministic automata*.

Often, these definitions also include a *start state* $s_0 \in \mathsf{State}_\mathsf{S}$ as part of the data. We don't do this.

Example 1.22. The Clock system can be seen as a deterministic system with:

$$\begin{pmatrix}\mathsf{tick}\\ \mathrm{id}\end{pmatrix} : \begin{pmatrix}\mathsf{Hour}\\ \mathsf{Hour}\end{pmatrix} \leftrightarrows \begin{pmatrix}\{*\}\\ \mathsf{Hour}\end{pmatrix}.$$

In other words, it consists of

- State set $\mathsf{State}_{\mathsf{Clock}} = \mathsf{Hour} = \{1, 2, \ldots, 12\}$.
- Output set $\mathsf{Out}_{\mathsf{Clock}} = \mathsf{Hour}$.
- Input set $\mathsf{In}_{\mathsf{Clock}} = \{*\}$, a one element set.
- Readout function $\mathrm{expose}_{\mathsf{Clock}} = \mathrm{id}_{\mathsf{Hour}}$.
- update function $\mathrm{update}_{\mathsf{Clock}} : \mathsf{hours} \times \{*\} \to \mathsf{Hour}$ defined by $\mathrm{update}_{\mathsf{Clock}}(t, *) = \mathsf{tick}(t)$.

Example 1.23. The term *Moore machine* is often used for the mathematical notion of deterministic system we've just presented, but it is also used for actual, real-life circuits which are designed on that principle.

For example, suppose that a wire carries the signals $\mathsf{Wire} = \{\texttt{high}, \texttt{low}\}$. We can see a deterministic system M with input $\mathsf{In}_{\mathsf{M}} = \mathsf{Wire}^n$ and $\mathsf{Out}_{\mathsf{M}} = \mathsf{Wire}^k$ as a circuit with n incoming wires and k outgoing wires.[a] The state then describes the state of all the internal wires (and capacitors, etc.) in the circuit. We would wire up these systems by literally wiring them together.

[a] Of course, the notion of "incoming" and "outgoing" wires are ways we think about the circuit in design terms. Circuits aren't actually directed in this way. We'll think about undirected notions of system in Chapter 2.

Note that when we say that a system doesn't have any parameters, as in Example 1.22, we don't take the parameter set to be empty but instead take it to have a single dummy value $\{*\}$, the one-element "hum of existence". In other words, having "no parameters" really means that the parameters are unchanging, or that there is no way to change the value of the parameters.

Also, we are just exposing the whole state with the system in Example 1.22. There is nothing preventing our systems from exposing their whole state, but often some aspects of the state are private, i.e. not exposed for use by other systems.

Exercise 1.24. Write out the clock and meridian systems from (1.6) and (1.9) in terms of Definition 1.20. Really, this amounts to noticing which sets are the sets of states, which are the sets of inputs, and what (implicitly) are the sets of outputs. ◊

Example 1.25 (SIR model). The set of states for a deterministic system doesn't need to be finite. The SIR model is an epimediological model used to study how a disease spreads through a population. "SIR" stands for "susceptible", "infected", and, rather ominously, "removed". This model is usually presented as a system of differential equations — what we will call a differential system — and we will see it in that form in **??**. But we can see a discrete approximation to this continuous model as a deterministic system.

A state of the SIR model is a choice of how many people are susceptible, how many are infected, and how many are removed. That is,

$$\mathsf{State}_{\mathsf{SIR}} = \left\{ \begin{bmatrix} s \\ i \\ r \end{bmatrix} \middle| \, s, \, i, \, r \in \mathbb{R} \right\} \cong \mathbb{R}^3.$$

is a 3-place vector of real numbers. We will again expose the whole state, so $\mathsf{Out}_{\mathsf{SIR}} = \mathsf{State}_{\mathsf{SIR}}$ and $\mathsf{expose}_{\mathsf{SIR}} = \mathrm{id}$.

The idea behind the SIR model is that if a susceptible person comes in contact with an infected person, then they have a chance of becoming infected too. And, eventually, infected persons will be removed from the model, either by recovering (a gentler way to read the "R") or by dying. So we need two parameters: the rate a of infection and the rate b of removal:

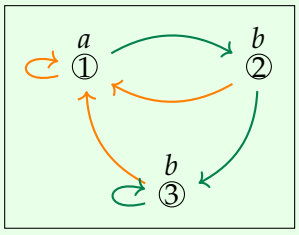
$$\mathsf{In}_{\mathsf{SIR}} = \left\{ \begin{bmatrix} a \\ b \end{bmatrix} \middle| \, a, \, b \in \mathbb{R} \right\} = \mathbb{R}^2.$$

Now, we can show how a population will develop according to this model by defining the update function:

$$\mathsf{update}_{\mathsf{SIR}} : \mathsf{State}_{\mathsf{SIR}} \times \mathsf{In}_{\mathsf{SIR}} \to \mathsf{State}_{\mathsf{SIR}} \tag{1.26}$$

$$\left(\begin{bmatrix} s \\ i \\ r \end{bmatrix}, \begin{bmatrix} a \\ b \end{bmatrix} \right) \mapsto \begin{bmatrix} s - asi \\ i + asi - bi \\ r + bi \end{bmatrix} \tag{1.27}$$

Example 1.28. If a deterministic system has a small finite set of states, then we can draw it entirely as a *transition diagram*:



Note that every node has an orange and a green arrow emanating from it, but that there are no rules on how many arrows point to it.

This diagram describes the following system S:

$$\begin{pmatrix}\text{update}_S\\ \text{expose}_S\end{pmatrix} : \begin{pmatrix}\{1,2,3\}\\ \{1,2,3\}\end{pmatrix} \leftrightarrows \begin{pmatrix}\{\texttt{green}, \texttt{orange}\}\\ \{a,b\}\end{pmatrix}.$$

That is, we have

- $\text{States}_S = \{1,2,3\}$.
- $\text{In}_S = \{\texttt{green}, \texttt{orange}\}$,
- $\text{Out}_S = \{a,b\}$,
- $$\begin{aligned}\text{expose}_S : \text{States}_S &\to \text{Out}_S\\ 1 &\mapsto a\\ 2 &\mapsto b\\ 3 &\mapsto b\end{aligned} \qquad \begin{aligned}\text{update}_S : \text{States}_S \times \text{In}_S &\to \text{In}_S\\ (1, \texttt{green}) &\mapsto 2\\ (1, \texttt{orange}) &\mapsto 1\\ (2, \texttt{green}) &\mapsto 3\\ (2, \texttt{orange}) &\mapsto 1\\ (3, \texttt{green}) &\mapsto 3\\ (3, \texttt{orange}) &\mapsto 1\end{aligned}$$

To draw a transition diagram of a system S, we draw each state $s \in \text{States}_S$ as a bubble filled with the label $\text{expose}_S(s)$, and for each parameter $i \in \text{In}_S$ we draw an arrow from s to $\text{update}_S(s,i)$. For a diagram like this to be a transition diagram, every node must have an edge leaving it for each parameter.

Exercise 1.29. Draw the Clock system (Example 1.22) as a transition diagram. ◊

Example 1.30 (Deterministic Finite Automata). A *deterministic finite automaton* (DFA) is a simple model of computation. Given our definition of deterministic system, DFAs are easy enough to define: they are just the deterministic systems with finitely many states whose output values are either `true` or `false`.

This means that the exposed variable of state $\text{expose}_S : \text{States}_S \to \{\texttt{true}, \texttt{false}\}$ is a boolean valued function. We say a state s is an *accept state* if $\text{expose}_S(s) = \texttt{true}$, and a *reject state* if $\text{expose}_S(s) = \texttt{false}$.

The idea is that a DFA is a question answering machine. Given a starting state s_0 and a sequence of input values $i_1, \ldots, i_n$, we get a sequence of states by $s_{t+1} := \text{update}_S(s_t, i_t)$. The answer to the question is "yes" if s_n is an accept state, and "no" if s_n is a reject state.

There is an important special case of deterministic systems which appear very commonly in the literature: the *closed* systems. These are the systems which have no

parameters, and which expose no variables. They are closed off from their environment, and can't be wired into any other systems.

As mentioned after Example 1.22, when we say "no" in this way — no parameters, no variables — we should be careful with what we mean exactly. We mean that there is no *variation* in the parameters or variables, that they are trivial. That is, we make the following definition.

Definition 1.31. A deterministic system S is *closed* if both In_S and Out_S have only one element

$$\mathsf{In}_\mathsf{S} \cong \{*\} \cong \mathsf{Out}_\mathsf{S}.$$

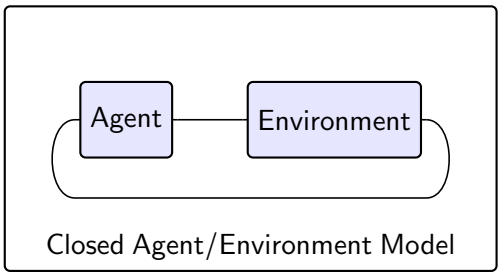
Exercise 1.32. Show that to give a closed system

$$\begin{pmatrix}\mathsf{update}_\mathsf{S}\\ \mathsf{expose}_\mathsf{S}\end{pmatrix} : \begin{pmatrix}\mathsf{State}_\mathsf{S}\\ \mathsf{State}_\mathsf{S}\end{pmatrix} \leftrightarrows \begin{pmatrix}\{*\}\\ \{*\}\end{pmatrix},$$

one just needs to choose a set $\mathsf{State}_\mathsf{S}$ and an update function $\mathsf{update}_\mathsf{S} : \mathsf{State}_\mathsf{S} \to \mathsf{State}_\mathsf{S}$. ◊

Given that we are mostly interested in how systems wire together, it may seem strange to draw attention to the closed systems that *can't* be wired into anything else. But we will often end up with a closed system as the result of wiring together some systems.

For example, suppose we have an Agent acting within a Environment. The agent will take an action, and the evironment will respond to that action. Depending on the action taken and response given, the agent and the environment will update their states. We can model this by the following wiring diagram:



To have this be a closed model is to think — or pretend — that the our model of the Agent and the Environment includes all possible external parameters, that it is well isolated from its own environment.

Exercise 1.33. What would happen to a system S if its set of parameters or output values were actually empty sets? Let's find out.

1. Suppose $\mathsf{In}_S = \emptyset$. Explain the content of a deterministic system

$$\begin{pmatrix}\mathsf{update}_S\\ \mathsf{expose}_S\end{pmatrix} : \begin{pmatrix}\mathsf{State}_S\\ \mathsf{State}_S\end{pmatrix} \leftrightarrows \begin{pmatrix}\emptyset\\ \{*\}\end{pmatrix}.$$

2. Suppose $\mathsf{Out}_S = \emptyset$. Explain the content of a deterministic system

$$\begin{pmatrix}\mathsf{update}_S\\ \mathsf{expose}_S\end{pmatrix} : \begin{pmatrix}\mathsf{State}_S\\ \mathsf{State}_S\end{pmatrix} \leftrightarrows \begin{pmatrix}\{*\}\\ \emptyset\end{pmatrix}. \qquad \lozenge$$

1.2.2 Differential systems

La nature ne fait jamais des sauts - Liebniz

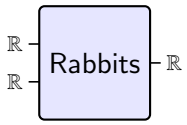
A quirk of modeling dynamical systems as determinstic systems is that determinstic systems lurch from one state to the next. In life, there are no next moments. Time, at least at human scales and to a first approximation, flows continuously.

Instead of modelling the "next" state a system will be in, we can model *how the system is tending to change*, in the moment. In order to do this, we need to make concession in the way we model the states of our system: we must assume they form a continuum themselves.

For example, suppose we are studying a population of Rabbits. We can measure the rate at which rabbits are born, and the rate they die. Then the population changes according to these rates. We can express this dependency of the change in population on certain rates with a differential equation:

$$\frac{dr}{dt} = \mathsf{b}_{\mathsf{Rabbits}} \cdot r - \mathsf{d}_{\mathsf{Rabbits}} \cdot r$$

where $r \in \mathbb{R}$ is the population of rabbits (considered as a real number for convenience), and the rates $\mathsf{b}_{\mathsf{Rabbits}}$ and $\mathsf{d}_{\mathsf{Rabbits}}$. The state of our system of Rabbits is the current population of rabbits, so $\mathsf{State}_{\mathsf{Rabbits}} = \mathbb{R}$, while we take the birth and death rates as parameters, so that $\mathsf{In}_{\mathsf{Rabbits}} = \mathbb{R} \times \mathbb{R}$. Accordingly, we can box the rabbit system up like so:



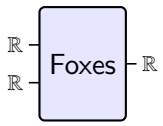
(1.34)

Now, rabbits are prey; they are eaten by other animals. That means that the rate at which rabbits die will depend on how often they are being eaten, and how often they are being eaten will depend on how many predators there are out there.

Now, the population of any predator will also change according to a birth rate and death rate. Suppose we have a similarly defined system of Foxes goverened whose population is governed by the differential equation

$$\frac{df}{dt} = \mathsf{b}_{\mathsf{Foxes}} \cdot f - \mathsf{d}_{\mathsf{Foxes}} \cdot f.$$

We can box up this system like so:



(1.35)

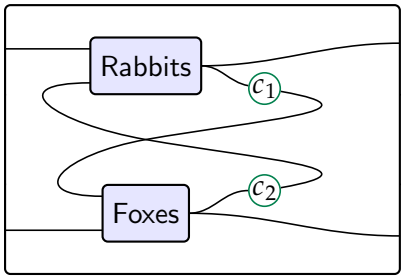
Now, we want the death rate of rabbits to depend on the number of foxes. But we also need the birth rate of the foxes to depend on the number of rabbits; after all, if a fox has nothing to eat, it has no time for hanky-panky. So we will add the following system of equations to the mix:

$$\begin{cases} \mathsf{d}_{\mathsf{rabbits}} = c_1 f \\ \mathsf{b}_{\mathsf{Foxes}} = c_2 r \end{cases}$$

Making these substitutions, we get the following system of differential equations:

$$\begin{cases} \frac{dr}{dt} = \mathsf{b}_{\mathsf{Rabbits}} \cdot r - c_1 f r \\ \frac{df}{dt} = c_2 r f - \mathsf{d}_{\mathsf{Foxes}} \cdot f \end{cases}$$

We are setting the parameters of the systems of Rabbits and Foxes according to the states of the other system. That is, we are wiring up the systems of Rabbits and Foxes:



(1.36)

The resulting system is called the *Lotka-Volterra predator-prey model*, and it is a simple differential model of the ways that the population of a predator species depends on the population of a prey species, and vice-versa.

Where before our boxes were filled with deterministic systems, now they are filled with systems of (first order, ordinary) differential equations. We call these *differential systems*.

Definition 1.37. A *(first order, ordinary) differential system* S with n state variables, m parameters, and k exposed variables

$$\begin{pmatrix} \text{update}_{\mathsf{S}} \\ \text{expose}_{\mathsf{S}} \end{pmatrix} : \begin{pmatrix} \mathbb{R}^n \\ \mathbb{R}^n \end{pmatrix} \leftrightarrows \begin{pmatrix} \mathbb{R}^m \\ \mathbb{R}^k \end{pmatrix}$$

consists of:

- An n-dimensional state space $\mathsf{State}_{\mathsf{S}} = \mathbb{R}^n$.
- An m-dimensional parameter space $\mathsf{In}_{\mathsf{S}} = \mathbb{R}^m$.
- A k-dimensional space of exposed variable values $\mathsf{Out}_{\mathsf{S}} = \mathbb{R}^k$.

- A smooth function $\text{update}_S : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ which gives us the derivative of each state variable at each time, so that the defining system of differential equations of S reads

$$\begin{cases} \frac{ds_1}{dt} &= \text{update}_{S1}(s, i) \\ &\vdots \\ \frac{ds_n}{dt} &= \text{update}_{Sn}(s, i). \end{cases}$$

- k exposed variables $\text{expose}_{Si} : \mathbb{R}^n \to \mathbb{R}$, which organize into a single smooth function $\text{expose}_S : \mathbb{R}^n \to \mathbb{R}^k$.

Remark 1.38. Definition 1.37 looks remarkably similar to Definition 1.20. As we mentioned, Definition 1.20 can be interpreted in any cartesian category, including the category **Euc** of Euclidean spaces and smooth maps (Definition 1.45). It appears that a differential system is the same thing as a deterministic system in the cartesian category **Euc**. But while the $\mathbb{R}^n$s occuring in $\text{update}_S : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ look the same, they are in fact playing very different roles. The $\mathbb{R}^n$ on the left is playing the role of the state space, while the $\mathbb{R}^n$ on the left is playing the role of the tangent space at s for some state $s \in \mathbb{R}^n$. The difference will be felt in Chapter 2 when we study behaviors of systems: the way a trajectory is defined for differential systems is different for differential systems and determinstic systems. For differential systems, a trajectory will be a solution to the system of differential equations, that is, a function $s : \mathbb{R} \to \mathbb{R}^n$ which satisfies

$$\frac{ds}{dt}(t) = \text{update}_S(s(t), i(t)).$$

for all choice of times t, while for a deterministic system a trajectory would be a sequence s_j of states so that $s_{j+1} = \text{update}_S(s_j, i_j)$.

We will see precisely how this difference is made manifest in the formal definition of a dynamical system doctrine as the choice of *section*.

Remark 1.39. There are other doctrines of differential systems that one can define (for example, allowing the state space to be a manifold), but in this book we will work with this simpler doctrine.

Example 1.40. The system of Rabbits has 1 state variable (the population of rabbits), 2 parameters (the birth and death rates of the rabbbits), and 1 exposed variable. It exposes its whole state, so that $\text{expose}_S = \text{id}$, and its update is given by

$$\text{update}_{\text{Rabbits}}(r, (\mathsf{b}_{\text{Rabbits}}, \mathsf{d}_{\text{Rabbits}})) = \mathsf{b}_{\text{Rabbits}} \cdot r - \mathsf{d}_{\text{Rabbits}} \cdot r.$$

The whole Lotka Voltera model of Eq. (1.36) has 2 state variables (the populations of rabbits and of foxes), 2 parameters (the birth rate of rabbits and the death rate of

foxes), and 2 exposed variables. It exposes its whole state, and its update is given by

$$\text{update}_{\mathsf{LK}}\left(\begin{bmatrix} r \\ f \end{bmatrix}, (\mathsf{b}_{\mathsf{Rabbits}}, \mathsf{d}_{\mathsf{Foxes}})\right) = \begin{bmatrix} \mathsf{b}_{\mathsf{Rabbits}} \cdot r - c_1 r f \\ c_2 f r - \mathsf{d}_{\mathsf{Foxes}} \cdot f \end{bmatrix}$$

One might wonder why we said this system has 2 parameters when there are also the rate constants c_1 and c_2 involved — aren't they also parameters? We chose them to be *constant*, where our parameters might vary over time. We could have made them parameters instead — it was an arbitrary choice in how to make the model.

Example 1.41. The most basic epidemiological model is the SIR model. This models the spread of disease through a population. People are either *susceptible* (S), *infected* (I), *recovered* or more ominously *removed* (R) from the model. When a susceptible person comes in contact with an infected person, they have a chance to become infected; this means that the population of susceptible people tends downwards in proportion to the number of susceptible and the number of infected people, and the population of infected people tends up by the same amount. On the other hand, infected people will eventually be removed from the model, either by recovering or dieing; this means that the population of infected people tends downwards proportional to the current infected population, while the removed population tends upwards by the same amount. Said as a system of differential equations, this means:

$$\begin{cases} \frac{dS}{dt} = -\alpha SI \\ \frac{dI}{dt} = \alpha SI - \beta I \\ \frac{dR}{dt} = \beta I \end{cases} \tag{1.42}$$

The SIR model is a differential system with 3 state variables (S, I, and R) and 2 parameters (α and β). We will suppose that it exposes its whole state: $\text{expose}_{\mathsf{SIR}} = \text{id}$. The update is given by

$$\text{update}_{\mathsf{SIR}}\left(\begin{bmatrix} S \\ I \\ R \end{bmatrix}, (\alpha, \beta)\right) = \begin{bmatrix} -\alpha SI \\ \alpha SI - \beta I \\ \beta I \end{bmatrix}.$$

In order to model higher order systems of ordinary differential equations, we will resort to the standard trick of encoding them as larger systems of first order systems. For example, to encode a second order differential equation in n variables, we would set the state space to be $\mathbb{R}^{2n}$ with state variables $(s, \dot{s})$ (the first n being s, the second n being $\dot{s}$). We then add the equations $\frac{ds}{dt} = \text{update}_{\mathsf{S}_s}((s, \dot{s}), i) := \dot{s}$ to the system $\frac{d\dot{s}}{dt} = \text{update}_{\mathsf{S}_{\dot{s}}}((s, \dot{s}), i)$ of second order differential equations we were trying to model.

Often, we want to think of the state variables $\dot{s}$ as *hidden* technical tricks. For this reason, we will often only expose the "actual" state variables s. This is one use for the function expose_S.

Example 1.43. Consider a mass m on a spring with a spring constant of c, taking position $s(t)$ at time t. Newton's second law then says that the acceleration of the mass is proportional to the force exerted upon it:

$$m\frac{d^2s}{dt} = -cs. \tag{1.44}$$

We can express this as a differential system in the following way. We take the state variables to be s and $\dot{s}$: $\mathsf{State}_{\mathsf{Spring}} := \mathbb{R}^2$. We will suppose that the mass and the spring constant are constant, so that this system takes no parameters: $\mathsf{In}_{\mathsf{Spring}} := \mathbb{R}^0 = \{*\}$. We will only expose the position of the spring, and not its velocity: $\mathsf{Out}_{\mathsf{Spring}} := \mathbb{R}$ and $\text{expose}_{\mathsf{Spring}}(s, \dot{s}) := s$. Finally, the dynamics of the system are given by:

$$\text{update}_{\mathsf{Spring}}\left(\begin{bmatrix} s \\ \dot{s} \end{bmatrix}\right) := \begin{bmatrix} \dot{s} \\ -\frac{cs}{m} \end{bmatrix}.$$

This is a way of re-writing Eq. (1.44) as a system of first order differential equations:

$$\begin{cases} \frac{ds}{dt} &= \dot{s} \\ \frac{d\dot{s}}{dt} &= -\frac{cs}{m} \end{cases}$$

Before we go on, we should clarify the category that we are working in when we work with our differential systems.

Definition 1.45. The category **Euc** is the category of *Euclidean spaces* and smooth maps between them. The objects of **Euc** are $\mathbb{R}^n$ for all $n \in \mathbb{N}$, and a morphism $f : \mathbb{R}^n \to \mathbb{R}^m$ is a smooth map.

We note that **Euc** is a cartesian category with $\mathbb{R}^n \times \mathbb{R}^m = \mathbb{R}^{n+m}$ and $1 = \mathbb{R}^0$.

1.3 Wiring together systems with lenses

In the last section, we saw the formal definition of deterministic and differential systems and a few examples of them. In this section, we'll see how to wire systems together — as we did in Section 1.1 for the clock and the universal Turing machine, and in Section 1.2.2 for the Lotka-Volterra predator prey model — to make more complex systems. We will do this using an interesting notion coming from the world of function programming: a *lens*.

1.3.1 Lenses and lens composition

A lens is a framework for bi-directional information passing.

Definition 1.46. A *lens* $\begin{pmatrix} f^\sharp \\ f \end{pmatrix} : \begin{pmatrix} A^- \\ A^+ \end{pmatrix} \leftrightarrows \begin{pmatrix} B^- \\ B^+ \end{pmatrix}$ in a cartesian category $\mathcal{C}$ consists of:

- A *passforward* map $f : A^+ \to B^+$, and
- a *passback* map $f^\sharp : A^+ \times B^- \to A^-$.

The most useful thing about lenses is that they *compose*.

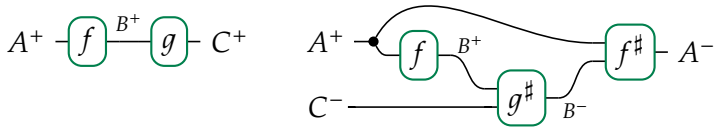
Definition 1.47. Let $\begin{pmatrix} f^\sharp \\ f \end{pmatrix} : \begin{pmatrix} A^- \\ A^+ \end{pmatrix} \leftrightarrows \begin{pmatrix} B^- \\ B^+ \end{pmatrix}$ and $\begin{pmatrix} g^\sharp \\ g \end{pmatrix} : \begin{pmatrix} B^- \\ B^+ \end{pmatrix} \leftrightarrows \begin{pmatrix} C^- \\ C^+ \end{pmatrix}$ be lenses in a cartesian category $\mathcal{C}$. We define their composite

$$\begin{pmatrix} g^\sharp \\ g \end{pmatrix} \circ \begin{pmatrix} f^\sharp \\ f \end{pmatrix}$$

to have passforward $g \circ f$ and passback

$$(a^+, c^-) \mapsto f^\sharp \left(a^+, g^\sharp(f(a^+), c^-) \right).$$

Here's a picture so that you can see the information flow for the composite of lenses:[1]



(1.48)

Remark 1.49. Even though our definition of lens was given in an arbitrary cartesian category $\mathcal{C}$, we felt comfortable defining it in terms of elements. Going forward, we will also reason with it using elements. This trick works for any cartesian category by using "generalized elements". We interpret an "element" x in an object X as a map $x : Z \to X$. If we do work with x to get a new element $f(x)$ of Y, then by the Yoneda lemma there is a map $f : X \to Y$ in the category which does that work by post-composition: $f(x) = f \circ x$.

The take-away is that even in a totally arbitrary cartesian category whose objects are not sets of any kind, we can still reason about them as if they were — at least when it comes to pairing elements and applying functions.

This gives us a category of lenses in any cartesian category $\mathcal{C}$.

[1] We draw this with a different style—green boxes, etc.—so that the reader will not confuse it with our usual wiring diagrams for systems. These are not dynamic in any way; everything below is just a set.

Definition 1.50. Let $\mathcal{C}$ be a cartesian category. Then the category $\mathbf{Lens}_{\mathcal{C}}$ has:

- as objects, the pairs $\binom{A^-}{A^+}$ of objects in $\mathcal{C}$, which we will call *arenas*.
- as morphisms, the lenses $\binom{f^\sharp}{f} : \binom{A^-}{A^+} \leftrightarrows \binom{B^-}{B^+}$.
- The identity lens is $\binom{\pi_2}{\mathrm{id}} : \binom{A^-}{A^+} \leftrightarrows \binom{A^-}{A^+}$, where $\pi_2 : A^+ \times A^- \to A^-$ is the projection.

Composition is given by lens composition as in Definition 1.47.

Remark 1.51. The category of lenses is special among categories because it is named for its *maps* (which are the lenses), rather than its objects (which are the arenas). This is because we will later meet another category, the *category of charts* (See Definition 2.34), whose objects are the arenas but whose maps are not lenses. Finally, in Definition 2.53 we will meet a *double category*[2] $\mathbf{Arena}_{\mathcal{C}}$ which combines these two categories whose objects are arenas and which *is* named after its objects. In **??**, we will explain the name "arena" and its role in the theory of dynamical systems.

Exercise 1.52.

1. Draw the composite of two lenses in the style of (1.48).
2. Check that $\mathbf{Lens}_{\mathcal{C}}$ is actually a category. That is, check that lens composition is associative, and that the identity lens is an identity for it. (Hint: You can use your drawing for this. You can slide the function beads around on the strings; if you pull a function bead past a split in the string, you have to duplicate it (since that split represents the duplication function).) ◊

Like any good categorical construction, $\mathbf{Lens}_{\mathcal{C}}$ varies functorially in its variable cartesian category $\mathcal{C}$.

Proposition 1.53 (Functoriality of **Lens**)**.** Every cartesian functor $F : \mathcal{C} \to \mathcal{D}$ induces a functor $\binom{F}{F} : \mathbf{Lens}_{\mathcal{C}} \to \mathbf{Lens}_{\mathcal{D}}$ given by

$$\binom{F}{F}\binom{f^\sharp}{f} = \binom{Ff^\sharp \circ \mu^{-1}}{Ff}$$

where $\mu = (F\pi_1, F\pi_2) : F(X \times Y) \xrightarrow{\sim} FX \times FY$ is the isomorphism witnessing that F preserves products.

Proof Sketch. Because lenses are defined just using the cartesian product, and F preserves these products, it commutes with everything in sight. □

[2] A double category is like a category with two different kinds of morphisms and a way for them to commute. See Definition 2.51 for the precise definition and the accompanying discussion.

Exercise 1.54.

1. What does the functor $\binom{F}{F} : \mathbf{Lens}_{\mathcal{C}} \to \mathbf{Lens}_{\mathcal{D}}$ do on objects?
2. Complete the proof of Proposition 1.53, by showing that $\binom{F}{F}$ really is a functor. ◊

1.3.2 Deterministic and differential systems as lenses

The reason we are interested in lenses and lens composition is because dynamical systems of various sorts are themselves lenses. As written in Definition 1.20, a system S is a lens in the category of sets of the form

$$\begin{pmatrix}\text{update}_{\mathsf{S}}\\ \text{expose}_{\mathsf{S}}\end{pmatrix} : \begin{pmatrix}\text{State}_{\mathsf{S}}\\ \text{State}_{\mathsf{S}}\end{pmatrix} \leftrightarrows \begin{pmatrix}\text{In}_{\mathsf{S}}\\ \text{Out}_{\mathsf{S}}\end{pmatrix}.$$

In fact, the deterministic systems are precisely the lenses whose input arena is of the form $\binom{S}{S}$. This means that we can compose a system S with a lens $\binom{f^\sharp}{f} : \binom{\text{In}_{\mathsf{S}}}{\text{Out}_{\mathsf{S}}} \leftrightarrows \binom{I}{O}$ to get a new dynamical system

$$\begin{pmatrix}\text{update}_{\mathsf{S}}\\ \text{expose}_{\mathsf{S}}\end{pmatrix} \fatsemi \begin{pmatrix}f^\sharp\\ f\end{pmatrix} : \begin{pmatrix}\text{State}_{\mathsf{S}}\\ \text{State}_{\mathsf{S}}\end{pmatrix} \leftrightarrows \begin{pmatrix}I\\ O\end{pmatrix}$$

with a new interface!

Similarly, a differential system is a lens in the category **Euc** of the form

$$\begin{pmatrix}\text{update}_{\mathsf{S}}\\ \text{expose}_{\mathsf{S}}\end{pmatrix} : \begin{pmatrix}\mathbb{R}^n\\ \mathbb{R}^n\end{pmatrix} \leftrightarrows \begin{pmatrix}\mathbb{R}^m\\ \mathbb{R}^k\end{pmatrix}.$$

We can then compose this with lenses in **Euc** to get new differential systems!

We can use this observation to wire together different systems. We separate this into to phases: first we put two systems in parallel, then we wire them together using a lens. The first phase, combine two systems without having them interact, is achieved through what we call the *parallel product* and denote $\otimes$. Two put two arenas $\binom{A_1}{B_1}$ and $\binom{A_2}{B_2}$ in parallel we just take their product in our cartesian category $\mathcal{C}$:

$$\begin{pmatrix}A_1\\ B_1\end{pmatrix} \otimes \begin{pmatrix}A_2\\ B_2\end{pmatrix} \coloneqq \begin{pmatrix}A_1 \times A_2\\ B_1 \times B_2\end{pmatrix}$$

In Definition 1.55 we define parallel product for morphisms in **Lens**, i.e. for general lenses.

Definition 1.55. For lenses $\binom{f^\sharp}{f} : \binom{A_1}{B_2} \leftrightarrows \binom{C_1}{D_1}$ and $\binom{g^\sharp}{g} : \binom{A_2}{B_2} \leftrightarrows \binom{C_2}{D_2}$, we define

their *parallel product*

$$\begin{pmatrix} f^\sharp \\ f \end{pmatrix} \otimes \begin{pmatrix} g^\sharp \\ g \end{pmatrix} : \begin{pmatrix} A_1 \times A_2 \\ B_1 \times B_2 \end{pmatrix} \leftrightarrows \begin{pmatrix} C_1 \times C_2 \\ D_1 \times D_2 \end{pmatrix}$$

to have passforward $f \times g$ and passback

$$((b_1, b_2), (c_1, c_2)) \mapsto (f^\sharp(b_1, c_2), g^\sharp(b_2, c_2)).$$

In terms of morphisms, this is

$$(B_1 \times B_2) \times (C_1 \times C_2) \xrightarrow{\sim} (B_1 \times C_1) \times (B_2 \times C_2) \xrightarrow{f^\sharp \times g^\sharp} A_1 \times A_2.$$

Together with $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$, this gives $\mathbf{Lens}_{\mathcal{C}}$ the structure of a monoidal category.

Remark 1.56. We will show a slick way to prove that the parallel product does indeed make $\mathbf{Lens}_{\mathcal{C}}$ into a monoidal category in Section 3.6.

Exercise 1.57. Show the parallel product of morphisms as in Definition 1.55 using the string diagram notation from (1.48). ◊

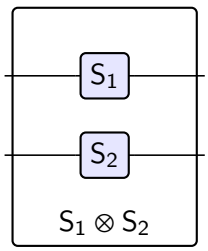
Proposition 1.58. Let $F : \mathcal{C} \to \mathcal{D}$ be a cartesian functor. The induced functor $\begin{pmatrix} F \\ F \end{pmatrix} : \mathbf{Lens}_{\mathcal{C}} \to \mathbf{Lens}_{\mathcal{D}}$ is strong monoidal with respect to the parallel product.

Proof. Since F preserves products, and . □

Given two dynamical systems S_1 and S_2, their parallel product $\mathsf{S}_1 \otimes \mathsf{S}_2$ is defined explicitly as follows:

- $\mathsf{State}_{\mathsf{S}_1 \otimes \mathsf{S}_2} := \mathsf{State}_{\mathsf{S}_1} \times \mathsf{State}_{\mathsf{S}_2}$.
- $\mathsf{Out}_{\mathsf{S}_1 \otimes \mathsf{S}_2} := \mathsf{Out}_{\mathsf{S}_1} \times \mathsf{Out}_{\mathsf{S}_2}$.
- $\mathsf{In}_{\mathsf{S}_1 \otimes \mathsf{S}_2} := \mathsf{In}_{\mathsf{S}_1} \times \mathsf{In}_{\mathsf{S}_2}$.
- $\mathrm{expose}_{\mathsf{S}_1 \otimes \mathsf{S}_2}((s_1, s_2)) = (\mathrm{expose}_{\mathsf{S}_1}(s_1), \mathrm{expose}_{\mathsf{S}_2}(s_2))$.
- $\mathrm{update}_{\mathsf{S}_1 \otimes \mathsf{S}_2}((s_1, s_2), (i_1, i_2)) = (\mathrm{update}_{\mathsf{S}_1}(s_1, i_1), \mathrm{update}_{\mathsf{S}_2}(s_2, i_2))$.

This can be expressed as the following wiring diagram:

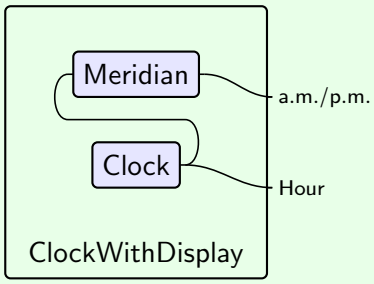


(1.59)

If we imagine physically wiring together our boxes, the first thing we would need to do is collect them together like this; then we can proceed to wire them. We will do

exactly this with our systems: first we will take their parallel product, and then we compose it with a lens that represents the wiring diagram.

Example 1.60. We can describe the ClockWithDisplay system (reproduced below) as a composite of lenses.



(1.61)

First, we take the parallel product of Meridian and Clock (see Exercise 1.24) to get the system

$$\text{Meridian} \otimes \text{Clock} : \begin{pmatrix} \text{a.m./p.m.} \times \text{Hour} \\ \text{a.m./p.m.} \times \text{Hour} \end{pmatrix} \leftrightarrows \begin{pmatrix} 1 \times \text{Hour} \\ \text{a.m./p.m.} \times \text{Hour} \end{pmatrix}.$$

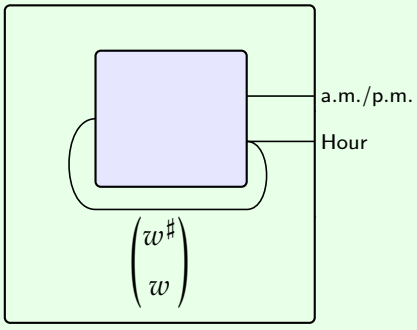
Now, we will express the wiring pattern in Eq. (1.61) as a lens

$$\begin{pmatrix} w^\sharp \\ w \end{pmatrix} : \begin{pmatrix} 1 \times \text{Hour} \\ \text{a.m./p.m.} \times \text{Hour} \end{pmatrix} \leftrightarrows \begin{pmatrix} 1 \\ \text{a.m./p.m.} \times \text{Hour} \end{pmatrix}.$$

We do this by setting

$$w(m, h) := (m, h), \text{ and}$$
$$w^\sharp((m, h), *) := (*, h).$$

Seen as a wiring diagram on its own, $\begin{pmatrix} w^\sharp \\ w \end{pmatrix}$ looks like this:

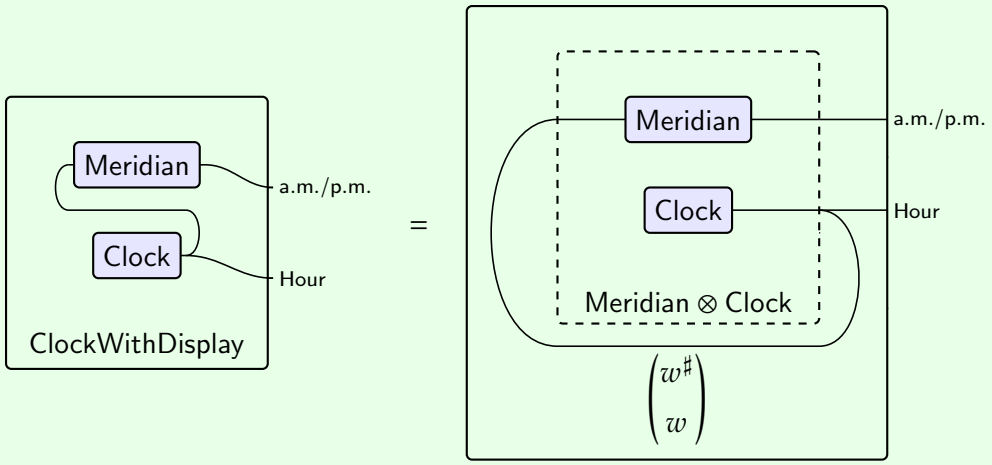


(1.62)

We can then see that

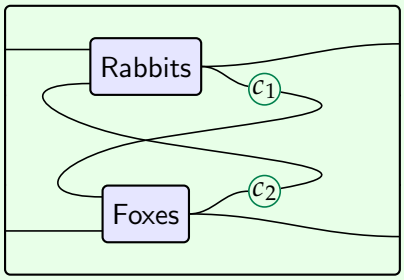
$$\text{ClockWithDisplay} = \begin{pmatrix} w^\sharp \\ w \end{pmatrix} \circ (\text{Meridian} \otimes \text{Clock})$$

just like we wanted! In terms of wiring diagrams, this looks like:



(1.63)

Example 1.64. We can describe the Lotka-Volterra predator prey model (reproduced below) as a composite of lenses.



(1.65)

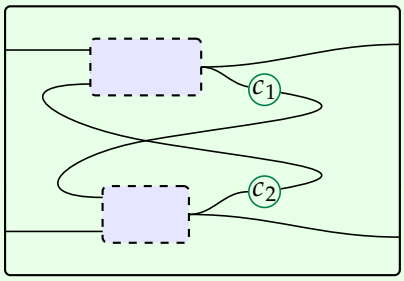
We can express the wiring pattern in **??** as a lens

$$\begin{pmatrix} w^\sharp \\ w \end{pmatrix} : \begin{pmatrix} \mathbb{R}^2 \\ \mathbb{R} \end{pmatrix} \otimes \begin{pmatrix} \mathbb{R}^2 \\ \mathbb{R} \end{pmatrix} \leftrightarrows \begin{pmatrix} \mathbb{R}^2 \\ \mathbb{R}^2 \end{pmatrix}.$$

We do this by setting

$$w(r, f) \coloneqq (r, f)$$
$$w^\sharp((r, f), (a, b)) \coloneqq (a, c_2 f, c_1 r, b)$$

We can draw $\begin{pmatrix} w^\sharp \\ w \end{pmatrix}$ as a wiring diagram on its own like this:



(1.66)

Filling those boxes with the systems of Rabbits and Foxes corresponds to taking the composite

$$(\mathsf{Rabbits} \otimes \mathsf{Foxes}) \fatsemi \begin{pmatrix} w^\sharp \\ w \end{pmatrix}$$

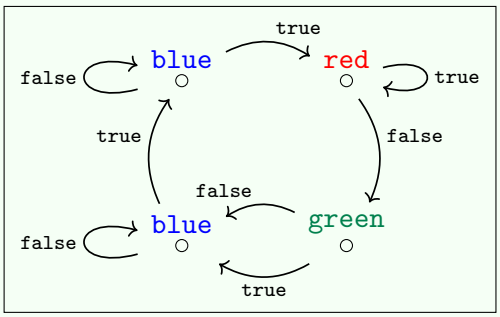
of lenses.

Wiring together transition diagrams. When a deterministic system is presented as a transition diagram (See Example 1.28), its dynamics is given by reading the input and following the arrow with that label, and then output the label on the resulting node. When we wire together systems presented as transition diagrams, the dynamics then involve reading the input labels of all inner systems, moving along all the arrows with those labels, and then outputing the labels at each state, possible into the input of another system.

Exercise 1.67. Here are two systems, S_1 and S_2 presented in terms of transition diagrams. The task is calculate the transition diagram of a system made by wiring them together.

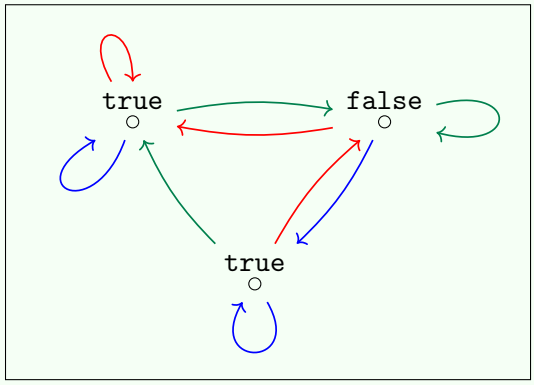
First, let Colors = {`red`, `blue`, `green`} and let Bool = {`true`, `false`}. Here is our first system S_1, which has interface $\begin{pmatrix} \mathsf{Bool} \\ \mathsf{Colors} \end{pmatrix}$:

$\mathsf{S}_1 :=$



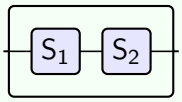
(1.68)

Our second system S_2 will have interface $\begin{pmatrix} \mathsf{Colors} \\ \mathsf{Bool} \end{pmatrix}$:

$S_2 :=$ 

(1.69)

1. Write down the transition diagram of the system obtained by connecting the above systems according to the following wiring diagram:

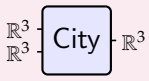
$S :=$ 

2. Explain how to understand the dynamics of this S in terms of the component systems S_1 and S_2. ◊

Multi-city SIR models In Examples 1.25 and 1.41, we saw deterministic and differential SIR models. Each models the spread of a disease through a single population. But what about a global pandemic where the disease is spreading through many local populations?

To model the spread of a disease through many different populations, we can use what is called a *multi-city SIR model*. We call each population a "city", and for now we will take flow of population between each city to be known constants. We can define a city as a differential system; then certain wiring diagrams of cities will correspond to multi-city models!

Definition 1.70. A City in a multi-city SIR model is a differential system



(1.71)

A city is defined by:

- $\mathsf{State}_{\mathsf{City}} := \left\{ \begin{bmatrix} S \\ I \\ R \end{bmatrix} \middle| S, I, R \in \mathbb{R} \right\} = \mathbb{R}^3.$
- $\mathsf{In}_{\mathsf{City}} = \{(\texttt{inflow}, \texttt{outflow}) \mid \texttt{inflow}, \texttt{outflow} \in \mathbb{R}^3\} = \mathbb{R}^3 \times \mathbb{R}^3$
- $\mathsf{Out}_{\mathsf{City}} = \mathsf{State}_{\mathsf{City}} = \mathbb{R}^3.$
- $\mathsf{expose}_{\mathsf{S}} = \mathrm{id}.$

- $$\text{update}_S\left(\begin{bmatrix} S \\ I \\ R \end{bmatrix}, (\texttt{inflow}, \texttt{outflow})\right) := \begin{bmatrix} -k_1 SI + \texttt{inflow}_1 - \texttt{outflow}_1 \\ k_1 SI - k_2 I + \texttt{inflow}_2 - \texttt{outflow}_2 \\ k_1 I + \texttt{inflow}_3 - \texttt{outflow}_3 \end{bmatrix}$$

  for some choice of constants k_1 and k_2.

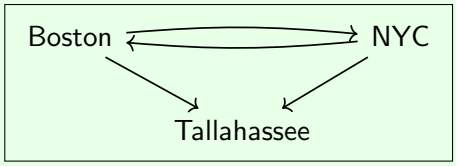
That is, each city will run its own SIR model, and each of the three populations can flow between cities.

Now, to define a mutli-city SIR model, we need to know what cities we are dealing with and how population flows between them. We'll call this a *population flow graph*.

Definition 1.72. A *population-flow graph* (for a mutli-city SIR model) is a graph whose nodes are labeled by cities and whose edges $\text{City}_1 \to \text{City}_2$ are labeled by 3×3 real diagonal matrices $\texttt{Flow}_{1\to 2}$ of the following form:

$$\begin{bmatrix} r_S & 0 & 0 \\ 0 & r_I & 0 \\ 0 & 0 & r_R \end{bmatrix}.$$

Example 1.73. Let's take a minute to understand Definition 1.72. Here is an example of a network of cities, represented in a graph:



(1.74)

This map contains three cities, Boston, NYC, and Tallahassee. As we can see, Boston and NYC have restricted access to travellers from Tallahassee, but otherwise people can travel freely. Let's focus in on one of these ways to travel, say Boston $\to$ NYC. This is associated to a matrix

$$\texttt{Flow}_{\text{Boston}\to\text{NYC}} := \begin{bmatrix} r_S & 0 & 0 \\ 0 & r_I & 0 \\ 0 & 0 & r_R \end{bmatrix}.$$

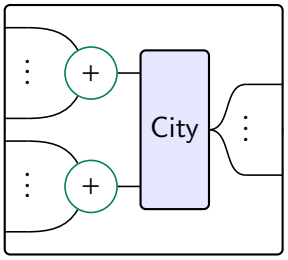
per the definition of a population flow graph. Here's how to understand this matrix. If the current population of Boston (split into susceptible, infected, and removed) is

$s = \begin{bmatrix} S \\ I \\ R \end{bmatrix}$, then

$$\texttt{Flow}_{\text{Boston}\to\text{NYC}}\, s = \begin{bmatrix} r_S & 0 & 0 \\ 0 & r_I & 0 \\ 0 & 0 & r_R \end{bmatrix} \begin{bmatrix} S \\ I \\ R \end{bmatrix} = \begin{bmatrix} r_S S \\ r_I I \\ r_R R \end{bmatrix}$$

is the population that will leave Boston and arrive in NYC. Of course, this assumes that people do not become sick in transit, a temporary assumption that a more robust model would have to address.

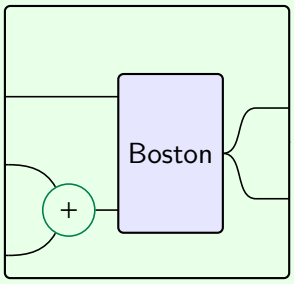
Given a population flow graph, we can form a multi-city SIR model by wiring together the cities in a particular way. Namely, to every city we will first add sums to its inputs for every city it is flowing to and every that flows to it. That is, we will prepare each city like so:



(1.75)

Specifically, we need to add together all the inflows from all other cities, and then record all the outflows to all other cities. We also need to copy the state enough times so that it can be passed to all other cities that our city flows to. So we need to add together inputs for all incoming edges in the population flow graph to the inflow port, and add together inputs for all outgoing edges in the population flow graph to the outflow port. And we also need to copy the output port to for all outgoing edges.

Example 1.76. For example, here is the preparation necessary for Boston in Eq. (1.74):

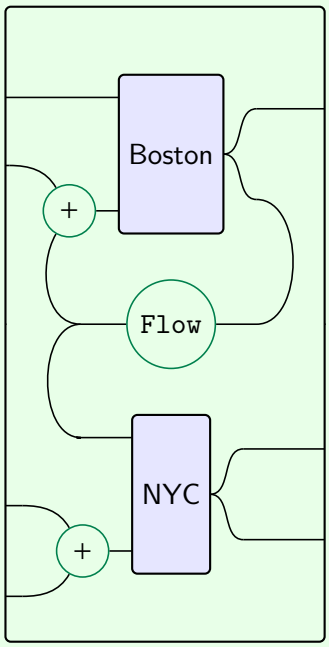


As you can see, there is only one incoming edge, and so the inflow input port doesn't need to anything to be added. But there are two outgoing edges, so we need to copy the output so they can be passed to NYC and Tallahassee and add together the two outflows into the outflow input port of Boston.

Exercise 1.77. Prepare the cities of NYC and Tallahassee from Eq. (1.74) in the same way Boston was prepared in Example 1.76. ◊

Next, we wire together these prepared cities (from Eq. (1.75)). For each edge $\text{City}_1 \to \text{City}_2$ in our population flow graph, we will put the matrix $\texttt{Flow}_{\text{City}_1 \to \text{City}_2}$ on the wire leaving the prepared City_1 corresponding to the edge, then split the wire and plug one end into the corresponding outflow input port of City_1 and the corresponding inflow input port of City_2.

Example 1.78. Here is what it looks like to wire Boston to NYC along the edge Boston → NYC in the population flow graph Eq. (1.74):



(1.79)

This wiring diagram says to take the population of Boston, take the proportion given by the flow rate $\texttt{Flow}_{\text{Boston} \to \text{NYC}}$, then set add this to the outflow parameter of Boston and the inflow parameter of NYC.

1.3.3 Wiring diagrams as lenses in categories of arities

We have been drawing a bunch of wiring diagrams so far, and we will continue to do so throughout the rest of the book. Its about time we explicitly described the rules one uses to draw these diagrams, and give a formal mathematical definition of them. The motto of this section is:

A wiring diagram is a lens in a free cartesian category — a category of arities.

We'll begin by describing wiring diagrams and their category in informal terms. Then, we will see how diagrams relate to lenses in a particular category — which we

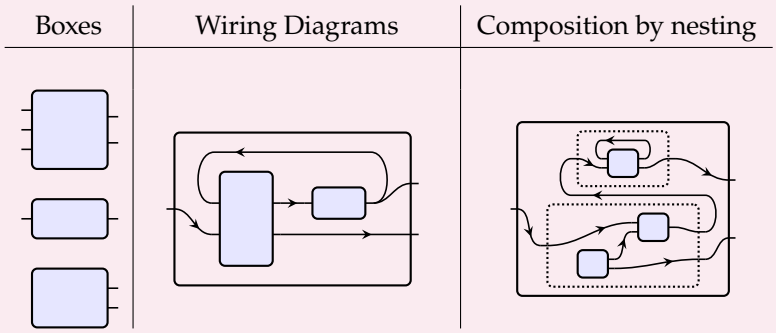
call the category of *arities* — and finally give a formal definition of the category of wiring diagrams.

Informal Definition 1.80. A *wiring diagram* is a diagram which consists of a number of *inner boxes*, each with some *input ports* and some *output ports*, that are wired together inside an *outer box*, which also has input and output ports. This gives four types of ports: inner (box) input (port), inner output, outer input, and outer output.

We can wire in the following ways:

1. Every outer output port is wired to exactly one inner output port.
2. Every inner input port is wired to exactly one inner output port or an outer input port.

The category of wiring diagrams has boxes as its objects and wiring diagrams as its morphisms. Wiring diagrams are composed by filling the inner boxes with other wiring diagrams, and then erasing the middle layer of boxes.



(1.81)

Wiring diagrams are designed to express the flow of variables through the system; how they are to be copied from one port to another, how they are to be shuffled about, and (though we haven't had need for this yet) how they are to be deleted or forgotton.

In order to capture this idea of copying, deleting, and shuffling around variables, we will work with the *category of arities* (and variations on it). The category of arities is extremely important since it captures precisely the algebra of copying, deleting, and shuffling around variables. In this section, we will interpret various sorts of wiring diagrams as lenses in categories of arities, which are the free cartesian categories.

Definition 1.82. The category **Arity** of arities is the free cartesian category generated by a single object X. That is, **Arity** constains an object X, called the *generic object*, and for any finite set I, there is an I-fold power X^I of X. The only maps are those that can be defined from the product structure by pairing and projection.

Explicitly, **Arity** is has:

- Objects $\{X^I \mid I \text{ a finite set}\}$.
- Maps $f^* : X^I \to X^J$ for any function $f : J \to I$.
- Composition defined by $g^* \circ f^* := (f \circ g)^*$ and $\mathrm{id} := \mathrm{id}^*$.

The cartesian product in **Arity** is given, in terms of index sets, by the following familiar

formula:

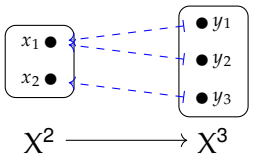
$$X^I \times X^J = X^{I+J}.$$

If you like opposite categories, this might clarify things a bit.

Proposition 1.83. **Arity** is isomorphic to the opposite of the category finite sets

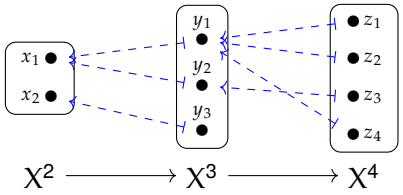
$$\mathbf{Arity} \cong \mathbf{FinSet}^{\mathrm{op}}.$$

Now, X is just a formal object, so it doesn't have elements. But we can give a language for writing down the objects and arrows of **Arity** that makes it look like it does. Think of the elements of X^I as finite lists of variables $X^I = (x_i \mid i \in I)$ indexed by the set I. Then for any reindexing function $f : J \to I$, we can see f^* as telling us how J-variables are assigned I-variables. For example, consider the function $f : 3 \to 2$ given by $1 \mapsto 2$, $2 \mapsto 1$, and $3 \mapsto 2$



In other words, f says that the first slot of the resulting list will be filled by the second variable of the first, and the second slot will be filled by the first variable, and the third slot will be filled by the second variable.

Composition is of course just given by composing functions in the opposite direction. For example, given some $g : 4 \to 3$, we just compose to get our map $X^2 \to X^4$.



Exercise 1.84. Express the following morphisms in **Arity** in terms of lists of variables:

1. The terminal morphism $X^2 \to X^0$, given by the *initial* function $!' : 0 \to 2$ which includes empty set into the set with two elements (hint, there's *nothing* on one side).
2. The duplication morphism $!^* : X \to X^2$ given by $! : 2 \to 1$.
3. The swap morphisms $\mathrm{swap}^* : X^2 \to X^2$ given by $\mathrm{swap} : 2 \to 2$ defined by $0 \mapsto 1$ and $1 \mapsto 0$.
4. What map corresponds to the map $1 : 1 \to 2$ picking out $1 \in 2 = \{1, 2\}$? What about $2 : 1 \to 2$.
5. Convince yourself that *any* map $X^I \to X^J$ you can express with the universal

property of products can be expressed by choosing an appropriate $f : J \to I$.

◊

Because **Arity** expresses the algebra of shuffling, copying, and deleting variables in the abstract, we can use it to define wiring diagrams. Recall from Definition 1.50 the definition of lens in an arbitrary cartesian category.

Definition 1.85. The category **WD** of wiring diagrams is defined to be the category of lenses in the category of arities **Arity**.

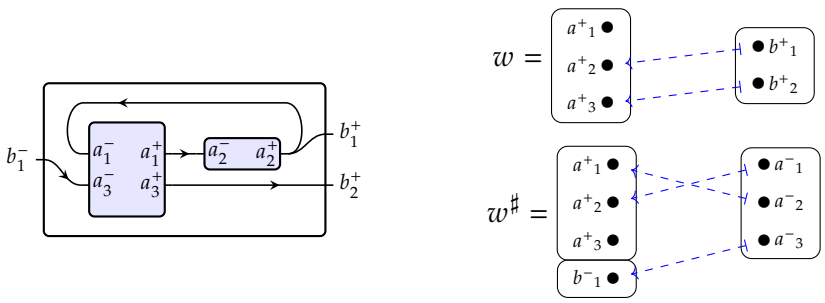
$$\mathbf{WD} := \mathbf{Lens}_{\mathbf{Arity}}.$$

We consider **WD** as a monoidal category in the same way we consider $\mathbf{Lens}_{\mathbf{Arity}}$ as a monoidal category.

This definition shows us that the wiring diagrams we have been using are precisely the lenses you can express if you only copy, delete, and shuffle around your variables. Here's how we interpret a lens $\begin{pmatrix} w^{\sharp *} \\ w^* \end{pmatrix} : \begin{pmatrix} \mathrm{X}^{A^-} \\ \mathrm{X}^{A^+} \end{pmatrix} \leftrightarrows \begin{pmatrix} \mathrm{X}^{B^-} \\ \mathrm{X}^{B^+} \end{pmatrix}$ in **Arity** as a wiring diagram:

- First, we interpret the index set A^- as the set of input ports of the inner boxes, and the set A^+ as the set of output ports of the inner boxes. Similarly, we see B^- as the set of input ports of the outer box, and B^+ as the set of output ports of the outer box.
- Then we remember that $w^* : \mathrm{X}^{A^+} \to \mathrm{X}^{B^+}$ comes from a reindexing function $w : B^+ \to A^+$), which we interpret as selecting for each outer output port $p \in B^+$, the unique inner output port $w(p)$ it will be wired to.
- Finally, we note that $w^{\sharp *} : \mathrm{X}^{A^+} \times \mathrm{X}^{B^-} \to \mathrm{X}^{A^-}$ comes from a function $w^\sharp : A^- \to A^+ + B^-$ (because $\mathrm{X}^{A^+} \times \mathrm{X}^{B^-} = \mathrm{X}^{A^+ + B^-}$), and we interpret this as selecting for each inner input port $p \in A^-$ either the inner output port $w^\sharp(p) \in A^+$ or the outer input port $w^\sharp(p) \in B^-$ which p will be wired to.

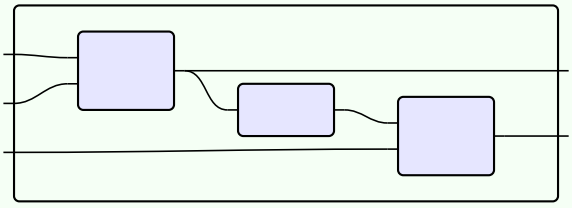
On the other hand, we can read any wiring diagram as a lens in **Arity** in the following way:



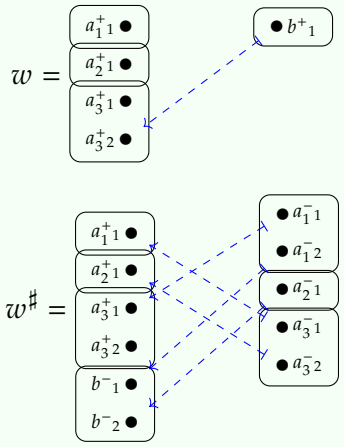
(1.86)

Exercise 1.87. Translate the following wiring diagrams into lenses in the category of arities, and vice versa:

1.

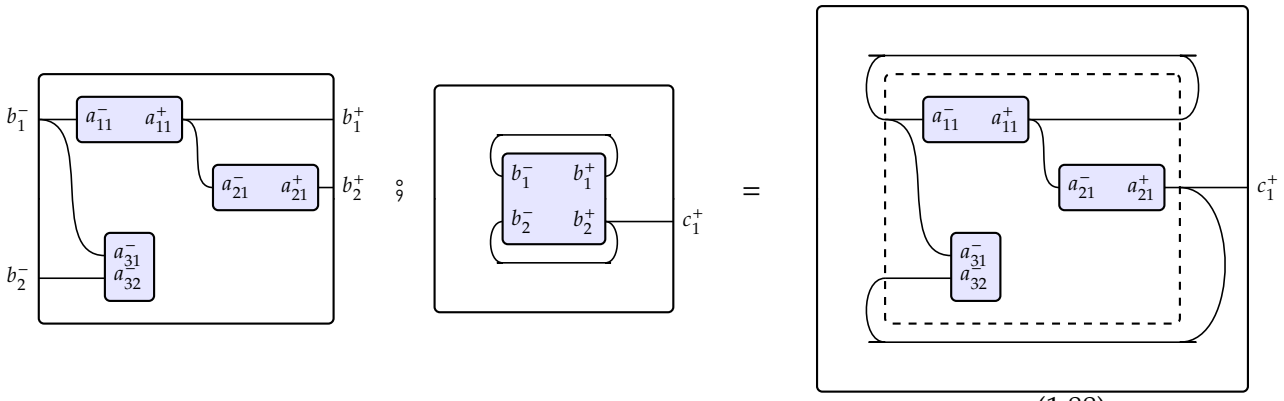


2. $\begin{pmatrix} w^\sharp \\ w \end{pmatrix} : \begin{pmatrix} X^2 \times X^1 \times X^2 \\ X \times X \times X^2 \end{pmatrix} \leftrightarrows \begin{pmatrix} X^2 \\ X^1 \end{pmatrix}$



◊

Ok, so the wiring diagrams correponds to the lenses in the category of arities. But do they compose in the same way? Composition of wiring diagrams is given by nesting: to compute $\begin{pmatrix} w^\sharp \\ w \end{pmatrix} \fatsemi \begin{pmatrix} u^\sharp \\ u \end{pmatrix}$, we fill in the inner box of $\begin{pmatrix} u^\sharp \\ u \end{pmatrix}$ with the outer box of $\begin{pmatrix} w^\sharp \\ w \end{pmatrix}$, and then remove this middle layer of boxes.



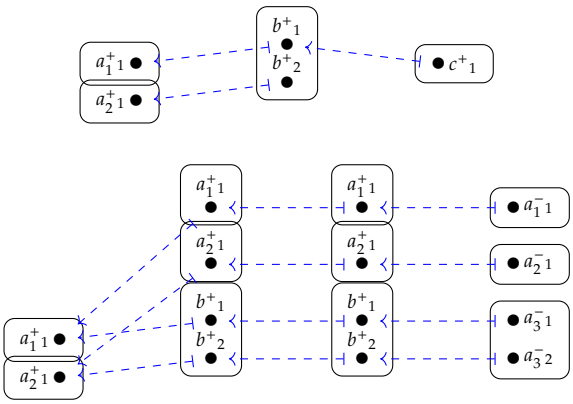
(1.88)

Let's say in prose how to compose two wiring diagrams. Then, we can check that this matches the formula given to us by lens composition in **Arity**.

- An outer output port is wired to a middle output port, and this middle output port is wired to an inner input port. So, to compose, we wire the outer output port to this inner output port.
- A inner input port is either wired to an inner input port or a middle input port. If it is wired to an inner input port, we leave it that way. Suppose that it was instead

wired to a middle input port. This middle input port is wired either to a middle output port or an outer input port. If it is wired to an outer input port, we then wire the inner input port to this outer input port. But if it was wired to a middle output port, we need to follow along to the inner output port that it is wired to; then we wire the inner input port to this inner output port.

Phew. After that block of text, I hope the mathematics will feel refreshingly crisp. Let's see what the lens composition looks like in **Arity**:



It's worth going through and seeing exactly how lens composition expresses the description we gave of nesting wiring diagrams above.

That **Arity** is the free cartesian category generated by a single object means that it satisfies a very useful universal property.

Proposition 1.89 (Universal property of **Arity**). For any cartesian category $\mathcal{C}$ and object $C \in \mathcal{C}$, there is a cartesian functor $\mathrm{ev}_C : \mathbf{Arity} \to \mathcal{C}$ which sends X to C. This functor is the unique such functor up to a unique natural isomorphism.

Proof Sketch. The functor ev_C can be defined by "just substitute C for X". Namely, we send

$$X^I \mapsto C^I$$

and for every function $f^* : X^I \to X^J$, we send it to $f^* : C^I \to C^J$ defined by the universal property of the product in $\mathcal{C}$. This is cartesian because $C^{I+J} \cong C^I \times C^J$ in any cartesian category. It is unique up to a unique natural isomorphism because X^I is the I-fold product of X, and so if $X \mapsto C$, then universal comparison maps between the image of X^I and C^I must be isomorphisms. □

We can think of the functor $\mathrm{ev}_C : \mathbf{Arity} \to \mathcal{C}$ as the functor which tells us how to interpret the abstract variables in **Arity** as variables of type C. For example, the functor $\mathrm{ev}_{\mathbb{R}} : \mathbf{Arity} \to \mathbf{Set}$ tells us how to interpret the abstract variables $(x_i \mid i \in I)$ in **Set** as variable real numbers $\{x_i \in \mathbb{R} \mid i \in I\}$. Under ev_C, the map of arities $(x_1, x_2, x_3 \mapsto x_2, x_2)$ gets sent to the actual map $C^3 \to C^2$ given by sending (c_1, c_2, c_3) to (c_2, c_2).

By the functoriality of the lens construction, this means that given an object $C \in \mathcal{C}$ of a cartesian category of "values that should be flowing on our wires", we can interpret a wiring diagram as a lens in $\mathcal{C}$! We record this observation in the following proposition.

Proposition 1.90. Let $C \in \mathcal{C}$ be an object of a cartesian category. Then there is a strong monoidal functor

$$\begin{pmatrix} \mathrm{ev}_C \\ \mathrm{ev}_C \end{pmatrix} : \mathbf{WD} \to \mathbf{Lens}_{\mathcal{C}}$$

which interprets a wiring diagram as a lens in $\mathcal{C}$ with values in C flowing along its wires.

Proof. This is just Proposition 1.53 (and **??**) applied to $\mathrm{ev}_C : \mathbf{Arity} \to \mathcal{C}$ from Proposition 1.89. □

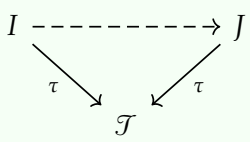
This is how we can interpret a wiring diagram as a lens in whatever cartesian category we are working in. There is, however, a slight issue; in most of our previous examples, there have been many different types of signals flowing along the wires. We can fix this by using *typed arities*. We will keep track of what type of signal is flowing along each wire, and only allow ourselves to connect wires that carry the same type of signal.

Definition 1.91. Let $\mathcal{T}$ be a set, elements of which we call *types*. The category $\mathbf{Arity}_{\mathcal{T}}$ is the free cartesian category generated by objects X_τ for each type $\tau \in \mathcal{T}$. Explicitly, $\mathbf{Arity}_{\mathcal{T}}$ has:

- Objects $\prod_{i \in I} X_{\tau_i}$ for any finite set I and *typing function* $\tau_{(-)} : I \to \mathcal{T}$. We interpret $\tau_i \in \mathcal{T}$ as the type of element $i \in I$.
- Maps $f^* : \prod_{i \in I} X_{\tau_i} \to \prod_{j \in J} X_{\tau_j}$ for any function $f : I \to J$ which preserves the typing: $\tau_{fi} = \tau_i$.
- Composition is given by $g^* \circ f^* = (f \circ g)^*$ and the identity is given by $\mathrm{id} := \mathrm{id}^*$.

That is, $\mathbf{Arity}_{\mathcal{T}} \cong (\mathbf{Fin} \downarrow \mathcal{T})^{\mathrm{op}}$ is dual to the category $\mathbf{Fin} \downarrow \mathcal{T}$ of *$\mathcal{T}$-typed finite sets*, the slice category (a.k.a. comma category) of the inclusion $\mathbf{Fin} \hookrightarrow \mathbf{Set}$ over the set $\mathcal{T}$ of types.

Exercise 1.92. We blew through that isomorphism $\mathbf{Arity}_{\mathcal{T}} \cong (\mathbf{Fin} \downarrow \mathcal{T})^{\mathrm{op}}$ quickly, but its not entirely trivial. The category $\mathbf{Fin} \downarrow \mathcal{T}$ has objects functions $\tau : I \to \mathcal{T}$ where I is a finite set, and a morphism is a commuting triangle like this:



Expand the isomorphism out in full and check that you understand it. ◊

Note that **Arity** = $\mathbf{Arity}_1$ is the special case where we have a single type. Just as we wrote the morphisms in **Arity** as $(x_1,\ x_2 \mapsto x_2,\ x_1,\ x_2)$, we can write the morphisms in $\mathbf{Arity}_{\mathcal{T}}$ as

$$(x_1 : \tau_1,\ x_2 : \tau_2,\ x_3 : \tau_2 \mapsto x_2 : \tau_2,\ x_1 : \tau_1,\ x_2 : \tau_1)$$

where $\tau_1,\ \tau_2,\ \tau_3 \in \mathcal{T}$ are all (fixed, not variable) types.

We check that $\mathbf{Arity}_{\mathcal{T}}$ as we defined it does indeed have the correct universal property.

Proposition 1.93. For any $\mathcal{T}$-indexed family of elements $C_{(-)} : \mathcal{T} \to \mathcal{C}$ in a cartesian category $\mathcal{C}$, there is a cartesian functor $\mathrm{ev}_C : \mathbf{Arity}_{\mathcal{T}} \to \mathcal{C}$ sending X_τ to C_τ. The functor ev_C is the unique such functor up to a unique natural isomorphism.

Proof Sketch. Just like in Proposition 1.89, we can take

$$\mathrm{ev}_C\left(\prod_{i\in I} X_{\tau_i}\right) := \prod_{i\in I} C_{\tau_i}.$$

□

Exercise 1.94. Complete the proof of Proposition 1.93. ◊

As before, we note that the map

$$(x_1 : \tau_1,\ x_2 : \tau_2,\ x_3 : \tau_2 \mapsto x_2 : \tau_2,\ x_1 : \tau_1,\ x_2 : \tau_1)$$

Spiz: redraw gets sent by ev_C to the function $C_{\tau_1} \times C_{\tau_2} \times C_{\tau_3} \to C_{\tau_2} \times C_{\tau_1} \times C_{\tau_2}$ which sends $(c_1,\ c_2,\ c_3)$ to $(c_2,\ c_1,\ c_2)$

Corollary 1.95. For any function $f : \mathcal{T} \to \mathcal{T}'$, there is a change of type functor $\mathrm{ev}_{X_f} : \mathbf{Arity}_{\mathcal{T}} \to \mathbf{Arity}_{\mathcal{T}}$.

Proof. We apply Proposition 1.93 to the family $X_{f(-)} : \mathcal{T} \to \mathbf{Arity}_{\mathcal{T}'}$ of objects of $\mathbf{Arity}_{\mathcal{T}'}$. That is, we send

$$\prod_{i\in I} X_{\tau_i} \mapsto \prod_{i\in I} X_{\tau(f(i))}.$$

□

We can now define the category of typed wiring diagrams to be the category of lenses in the category of typed arities.

Definition 1.96. For a set $\mathcal{T}$ of types, the category $\mathbf{WD}_{\mathcal{T}}$ of *$\mathcal{T}$-typed wiring diagrams* is

the category of lenses in the category of $\mathcal{T}$-typed arities:

$$\mathbf{WD}_{\mathcal{T}} := \mathbf{Lens}_{\mathcal{T}}.$$

As with the singly-typed case, we can interpret any typed wiring diagram as a lens in a cartesian category of our choosing.

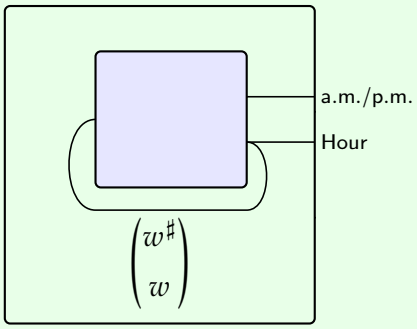
Proposition 1.97. For any family $C_{(-)} : \mathcal{T} \to \mathcal{C}$ of objects in a cartesian category $\mathcal{C}$, indexed by a set $\mathcal{T}$ of types, there is a strong monoidal functor

$$\begin{pmatrix} \mathrm{ev}_C \\ \mathrm{ev}_C \end{pmatrix} : \mathbf{WD}_{\mathcal{T}} \to \mathbf{Lens}_{\mathcal{C}}$$

which interprets a typed wiring diagram as a lens in $\mathcal{C}$ with appropriately typed values flowing along its wires.

Remark 1.98. Because the action of ev_C is so simple, we will often just equate the typed wiring diagram with the lens it gives when interpreted in our category of choice.

Example 1.99. We can describe the wiring diagram



(1.100)

from Example 1.60 as a lens in a category of typed arities in the following way. We have two types: a.m./p.m. and Hour. So, $\mathcal{T} = \{\text{a.m./p.m.}, \text{Hour}\}$. Then

$$w = (t : \text{Hour}, m : \text{a.m./p.m.} \mapsto t : \text{Hour}, m : \text{a.m./p.m.})$$
$$w^\sharp = (t : \text{Hour}, m : \text{a.m./p.m.} \mapsto t : \text{Hour})$$

giving us a wiring diagram in $\mathbf{WD}_{\mathcal{T}}$. We can then interepret this as the lens from *Example* 1.60 as the image of this wiring diagram which interprets the types a.m./p.m. and Hour as the actual sets $\{\texttt{a.m.}, \texttt{p.m.}\}$ and $\{1, 2, \ldots, 12\}$.

1.3.4 Wiring diagrams with operations as lenses in Lawvere theories

The wiring diagrams we have described as lenses in categories of arities are pure wiring diagrams. But in Example 1.64, we used a wiring diagram (Eq. (1.66)) with little

green beads representing multiplication by a constant scalar, and in Section 1.3.2 we used a wiring diagram with little green beads representing multiplication by a matrix (Eq. (1.79)). It is very useful to be able to perform operations on the exposed variables we are passing to parameters.

In this section, we will see that if we have an *algebraic theory* of the kinds of operations we want to perform on our variables while we wire them, we can describe wiring diagrams with green beads representing those adjustments as lenses in the *Lawvere theory* of that algebraic theory.

Algebraic theories are theories of operations that are subject to certain equational laws.

Informal Definition 1.101. A *algebraic theory* $\mathbb{T}$ consists:

- A set $\mathbb{T}_n$ of n-ary operations for each $n \in \mathbb{N}$.
- A set of *laws* setting some composites of operations equal to others.

Example 1.102. The algebraic theory of real vector spaces can be described like this:

- There is a binary operation $(-) + (-)$ of vector addition, and for every $r \in \mathbb{R}$ a unary operation $r \cdot (-)$ of scalar multiplication, and a nullary operation (a.k.a. constant) 0.
- These satisfy the laws that make $+$ and 0 into an abelian group with addition inverses given by $-1 \cdot (-)$, and which satisfy associativity and distributivity with regards to scalar multiplication.

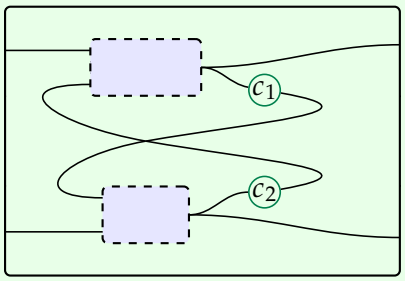
$$\begin{aligned} (a+b)+c &= a+(b+c) & r \cdot (s \cdot a) &= (rs) \cdot a \\ 0 + a &= a & (r+s) \cdot a &= r \cdot a + s \cdot a \\ a + b &= b + a & 1 \cdot a &= a \\ a + (-1 \cdot a) &= 0 & 0 \cdot a &= 0 \end{aligned}$$

We can use an algebraic theory to organize the sorts of operations we are willing or able to perform on the values flowing through the wires of our wiring diagrams.

Informal Definition 1.103. A wiring diagram with operations from an algebraic theory $\mathbb{T}$ is a wiring diagram where operations from the theory $\mathbb{T}$ can be drawn in little green beads on the wires.

Example 1.104. The wiring diagram Eq. (1.66) (reproduced below) is a wiring diagram in the algebraic theory of real vector spaces. The little green beads have scalar multi-

plications drawn in them.



We want to make these informal definitions precise. Ultimately, we want to be able to say that "wiring diagrams with operations from $\mathbb{T}$ are lenses in such and such cartesian category". We can do this with the notion of Lawvere theory.

Lawvere introduced his theories in his 1963 thesis "Functorial Semantics of Algebraic Theories" [] as the invariant concepts of algebraic theories, freed from any particular presentation by symbols and their relations. In Example 1.102, we presented the algebraic theory of real vector spaces in a particular way; but we could have done it differently, say by avoiding the vector 0 entirely and adding the law $(0 \cdot a) + b = b$. Lawvere wanted to avoid these petty differences in presentation. He focuses instead on the cartesian category freely containing the operations of the theory (satisfying their laws). This gives an invariant of the concept of real vector space that is independent of how that concept is presented.

A Lawvere theory is, in some sense, a category of arities "with extra maps". We think of these extra maps as coming from the operations of some theory.

Definition 1.105. A $\mathcal{T}$-sorted Lawvere theory $\mathcal{L}$ is a cartesian category equipped with a bijective-on-objects functor $\mathbf{Arity}_{\mathcal{T}} \hookrightarrow \mathcal{L}$.

If $\mathcal{T}$ has a single element, we refer to this as a *single sorted* Lawvere theory.

Where we wrote the objects of **Arity** as X^I to suggest the genericness of the generating object X, we will see that the objects of Lawvere theories are often A^I for some "actual" object A in some cartesian category.

Example 1.106. The single sorted Lawvere theory **Vect** of real vector spaces can be defined as follows:

- For every finite set I, it has an object $\mathbb{R}^I \in \mathbf{Vect}$.
- A map $f : \mathbb{R}^I \to \mathbb{R}^J$ is a linear map, or equivalently a $J \times I$ matrix.
- The cartesian product $\mathbb{R}^I \times \mathbb{R}^J$ is $\mathbb{R}^{I+J}$.

Since **Vect** is a cartesian category, it admits a functor $X \mapsto \mathbb{R}$ from **Arity**. By construction, this functor is bijective on objects; we just need to show that it is faithful. If $g^*, f^* : X^I \to X^J$ are such that $g^* = f^*$ as maps $\mathbb{R}^I \to \mathbb{R}^J$, then in particular $g^*(e_i) = f^*(e_i)$

for all standard basis vectors e_i defined by

$$e_i(j) \coloneqq \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

But $g^*(e_i)(j) \coloneqq e_i(g(j))$ and $f^*(e_i)(j) \coloneqq e_i(f(j))$, so by varying i we can test that $g(j) = f(j)$ for all $j \in J$, and therefore that $g^* = f^*$ as maps $X^I \to X^J$.

How do we know that the extra maps in a Lawvere theory really do come from the operations of an algebraic theory? We show that the Lawvere theory satisfies a certain universal property: cartesian functors out of it correpond to *models* of the theory. If this is the case, we say that the Lawvere theory is *presented* by the algebraic theory.

Informal Definition 1.107. Let $\mathbb{T}$ be an algebraic theory. A *model* of $\mathbb{T}$ in a cartesian category $\mathcal{C}$ is an object $C \in \mathcal{C}$ together with maps $m(f) : C^n \to C$ for each n-ary operation $f \in \mathbb{T}_n$ such that the maps $m(f)$ satisfy the laws of the theory.

Definition 1.108. A *model* of a Lawvere theory $\mathcal{L}$ in a cartesian category $\mathcal{C}$ is a cartesian functor $M : \mathcal{L} \to \mathcal{C}$.

We say that a Lawvere theory is presented by an algebraic theory if they have the same models in any cartesian category. We can show that our Lawvere theory **Vect** of vector spaces is presented by the theory of vector spaces of Example 1.102.

Proposition 1.109. Let $\mathcal{C}$ be a cartesian category. Then for every real vector space in $\mathcal{C}$, by which we mean an object $V \in \mathcal{C}$ with a binary addition $+ : V^2 \to V$, a unary scalar multiplication $r\cdot : V \to V$ for each $r \in \mathbb{R}$, and a nullary $0 : 1 \to V$ which satisfy the laws of a vector space, there is a cartesian functor $\hat{V} : \mathbf{Vect} \to \mathcal{C}$ sending $\mathbb{R}$ to V. Moreover, this functor is unique up to a unique isomorphism among functors sending $\mathbb{R}$ to V.

Proof Sketch. We define the functor $\hat{V}$ by sending $\mathbb{R}^I$ to V^I, and sending the operations $+ : \mathbb{R}^2 \to \mathbb{R}$, $r\cdot : \mathbb{R} \to \mathbb{R}$, and $0 : \mathbb{R}^0 \to \mathbb{R}$ to the corresponding operations on V. Given a general linear map $f : \mathbb{R}^I \to \mathbb{R}^J$, f can be expressed as a composite of these operations; therefore, we can define $\hat{V}(f)$ to be the corresponding composite of the operations on V. □

Definition 1.110. Let $\mathcal{L}$ be a Lawvere theory. The category $\mathbf{WD}^{\mathcal{L}}$ of wiring diagrams with operations from $\mathcal{L}$ is the category of lenses in $\mathcal{L}$:

$$\mathbf{WD}_{\mathcal{L}} \coloneqq \mathbf{Lens}_{\mathcal{L}}.$$

Remark 1.111. The bijective-on-objects functor $\mathbf{Arity}_{\mathcal{T}} \to \mathcal{L}$ lets us interpret every $\mathcal{T}$-typed wiring diagram as a wiring diagram with operations from $\mathcal{L}$ by Proposition 1.97.

In order to interpret a wiring diagram with operations from $\mathcal{L}$ as a lens in a cartesian category $\mathcal{C}$, we need a cartesian functor $\mathcal{L} \to \mathcal{C}$. These are precisely the models of the Lawvere theory. So, if our interpretation of the wires of our diagrams are models of our Lawvere theory $\mathcal{L}$, we can interpret diagrams with operations from $\mathcal{L}$.

Example 1.112. The wiring diagram Eq. (1.79) is a wiring diagram with operations from **Vect**, the theory of vector spaces. This is why we are able to put matrices in the beads.

Chapter 2

Non-deterministic doctrines

So far, we have seen how deterministic systems of the discrete- and continuous-time variety can be wired together. But modelling a system deterministically can be a bit hubristic: it assumes we have taken account of all variables that act on the state of the system, so that we can know exactly what will happen next or exactly how the system is tending to change. Often we know that the way we've modeled state is incomplete, and so knowing the state in our model might not tell us exactly what will happen next.

As an example, consider a person typing out an email. We know that the output of this system over time will be a stream of ASCII characters, and we won't model the various sorts of inputs that might be affecting the person as they write the email. The particular email written will depend on the person's state, but this state is extraordinarily complex and modelling it to the point that we would know exactly which email they will write is nigh impossible.

So, instead, we could use what we know about how emails that this person writes tend to look to predict what the next character will be. This would give us a *stochastic* model of the email-writer system.

In this section, we will see a variety of non-deterministic (discrete-time) doctrines. The kind of non-determinism — possibilistic, stochastic, etc. — will be encoded in a *commutative monad* (Definition 2.7).

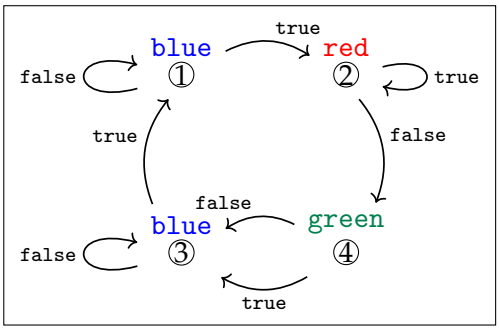
2.1 Possibilistic systems

Suppose that we are observing a deterministic system S from the outside. We can choose what input $i \in \mathsf{In}_{\mathsf{S}}$ to put into the system, and we observe from that what output $o \in \mathsf{Out}_{\mathsf{S}}$ comes out as a result. Can we understand how the system works from knowing this alone? In other words, can we construct a new system S' just from knowing how inputs relate to outputs in S?

In full generality, the answer is of course "no"; if there was only one possible output, for example, we have no chance to understand what's going on inside the system. But if we do observe a bunch of different changing outputs, we can give it a shot.

As a first guess, we might try to model how an input $i \in \mathsf{In}_\mathsf{S}$ changes the output $o \in \mathsf{Out}_\mathsf{S}$ that we are seeing. That is, we might try and make $\mathsf{States}_{\mathsf{S}'} = \mathsf{Out}_\mathsf{S}$, and then define the new dynamics $\mathrm{update}_{\mathsf{S}'}(o, i)$ be the new output S gives when fed input i while it is exposing output o. There's just one problem with this idea: we won't always get the same output when we feed i in to S while it's exposing o.

For example, consider the following transition system:

$\mathsf{S} :=$ 

(2.1)

The inputs to this system are from the set $\mathsf{In}_\mathsf{S} = \{\texttt{true}, \texttt{false}\}$, and the outputs are from the set $\mathsf{Out}_\mathsf{S} = \{\texttt{red}, \texttt{blue}, \texttt{green}\}$. Suppose that we can only see what the system is outputing, and that it is outputing `blue`. If we feed the system `false`, we will see `blue`. But, if we feed the system `true`, what happens depends on whether the system was in state 1 or state 3; if we were in state 1, then we will see `red`, but if were were in state 3, we will see `blue`. So, the next output is not uniquely determined by the current output and current input — there are many possibilities. We are tempted to say that `blue` will transition to *either* `red` or `blue` in our model S′ of the system S. That is, we want the update of S′ to tell us what is *possible*, since we can't know just from the outputs of S what is *determined* to happen. We can do that by having the update of S′ give us the set of possibilities:

$$\mathrm{update}_{\mathsf{S}'}(\texttt{blue}, \texttt{true}) = \{\texttt{blue}, \texttt{red}\}.$$

In this section, we will see two dynamical system doctrines which, instead of telling us the next state, tell us which states are possible or which are probable. Both are examples of *non-deterministic* doctrines, since the current state doesn't determine precisely the next state.

Definition 2.2. A *possibilistic system* S, also written as

$$\begin{pmatrix}\mathrm{update}_\mathsf{S} \\ \mathrm{expose}_\mathsf{S}\end{pmatrix} : \begin{pmatrix}\mathsf{States}_\mathsf{S} \\ \mathsf{States}_\mathsf{S}\end{pmatrix} \leftrightarrows \begin{pmatrix}\mathsf{In}_\mathsf{S} \\ \mathsf{Out}_\mathsf{S}\end{pmatrix},$$

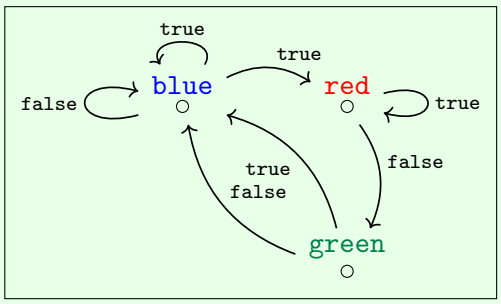
consists of:

- a set $\mathsf{States}_\mathsf{S}$ of *states*;
- a set Out_S of *values for exposed variables*, or *outputs* for short;
- a set In_S of *parameter values*, or *inputs* for short;

- a function $\text{expose}_S : \text{States} \to \text{Outs}$, the *exposed variable of state* or *expose* function, which takes a state to the output it yields; and
- a function $\text{update}_S : \text{States} \times \text{Ins} \to \text{PStates}$, where PStates_S is the set of subsets of States_S. This is the *dynamics* or *update* function which takes a state and a parameter and gives the set of possible next states.

Remark 2.3. While Definition 1.20 can be interepreted in any cartesian category because it only used maps and the cartesian product, Definition 2.2 makes use of the *power set* operation P which sends a set to its set of subsets. This can't be interpreted in any cartesian category — we need something resembling P in order for it to make sense.

Example 2.4. A possibilistic automaton can be presented as a transition diagram as well. Consider, for example, the following diagram:

$S' :=$ 

(2.5)

This system resembles system S of Eq. (2.1), except that it has a single state for each output. We can tell that this transition diagram represents a possibilistic system because there are two arrows leaving `blue` both labeled `true`. Since the dynamics of a transition diagram are given by following the arrow labeled by the input along to a new state, we see that here we will end up at a set of states:

$$\text{update}_{S'}(\texttt{blue}, \texttt{true}) = \{\texttt{blue}, \texttt{red}\}.$$

Example 2.6. In Example 1.30, we saw that deterministic finite automata (DFAs) are examples of deterministic systems. There is another common notion in automata theory: *non-deterministic* finite automata (NFAs).

An NFA is a possibilistic system S with finitely many states whose output values are Booleans: $\text{Outs} = \{\texttt{true}, \texttt{false}\}$. As with DFAs, the exposed variable $\text{expose}_S : \text{States} \to \{\texttt{true}, \texttt{false}\}$ tells us whether or not a state is an accept state.

Again, NFAs are question answering machines. But this time, since they are non-deterministic, we ask whether or not it is *possible* to accept a given sequence of inputs. Suppose we have a sequence of inputs $i_0, \ldots, i_n$, and we start in a state s_0. Now, because an NFA is possibilistic, we don't have a "next state" s_1. Rather, we have a set of states $S_1 := \text{update}_S(s_0, i_0)$. Now, we need to interatively define the next evolution: S_2 should

be the set of states that are possible to get to from *any* state in S_1. Generally,

$$S_{j+1} := \{s' \mid s \in S_j, s' \in \mathrm{update}_{\mathsf{S}}(s, i_j)\} = \bigcup_{s \in S_j} \mathrm{update}_{\mathsf{S}}(s, i_j)$$

We then say that the machine accepts the input sequence if there is any accept state in S_n.

Example 2.6 contains an answer to an interesting question: how do we iterate the behavior of a possibilistic system? For a deterministic system whose update has the signature $\mathrm{update}_{\mathsf{S}} : \mathsf{State}_{\mathsf{S}} \times \mathsf{In}_{\mathsf{S}} \to \mathsf{State}_{\mathsf{S}}$, we can compose to get

$$\mathsf{State}_{\mathsf{S}} \times \mathsf{In}_{\mathsf{S}} \times \mathsf{In}_{\mathsf{S}} \xrightarrow{\mathrm{update}_{\mathsf{S}} \times \mathsf{In}_{\mathsf{S}}} \mathsf{State}_{\mathsf{S}} \times \mathsf{In}_{\mathsf{S}} \xrightarrow{\mathrm{update}_{\mathsf{S}}} \mathsf{State}_{\mathsf{S}}$$

which sends $(s, (i_0, i_1))$ to $\mathrm{update}_{\mathsf{S}}(\mathrm{update}_{\mathsf{S}}(s, i_0), i_1)$. We can do this as many times as we like to apply an entire sequence of inputs to a state.

But for a possibilistic system, the update has signature $\mathrm{update}_{\mathsf{S}} : \mathsf{State}_{\mathsf{S}} \times \mathsf{In}_{\mathsf{S}} \to \mathsf{PState}_{\mathsf{S}}$. Now we can't just compose, if we tried the trick above we would go from $\mathsf{State}_{\mathsf{S}} \times \mathsf{In}_{\mathsf{S}} \times \mathsf{In}_{\mathsf{S}} \to \mathsf{PState}_{\mathsf{S}} \times \mathsf{In}_{\mathsf{S}}$, and we're stuck.

But from $\mathrm{update}_{\mathsf{S}} : \mathsf{State}_{\mathsf{S}} \times \mathsf{In}_{\mathsf{S}} \to \mathsf{PState}_{\mathsf{S}}$ we can define a function $U : \mathsf{PState}_{\mathsf{S}} \times \mathsf{In}_{\mathsf{S}} \to \mathsf{PState}_{\mathsf{S}}$ by

$$U(S, i) = \{s' \mid s \in S,\ s' \in \mathrm{update}_{\mathsf{S}}(s, i)\} = \bigcup_{s \in S} \mathrm{update}_{\mathsf{S}}(s, i)$$

Then we can define the iterated action of the system to be the composite

$$\mathsf{State}_{\mathsf{S}} \times \mathsf{In}_{\mathsf{S}} \times \mathsf{In}_{\mathsf{S}} \xrightarrow{\mathrm{update}_{\mathsf{S}}} \mathsf{PState}_{\mathsf{S}}\mathsf{In}_{\mathsf{S}} \xrightarrow{U} \mathsf{PState}_{\mathsf{S}}.$$

This process of lifting a function $A \times B \to \mathrm{P}C$ to a function $\mathrm{P}A \times B \to \mathrm{P}C$ is fundamental, and worthy of abstraction. This operation comes from the fact that P is a *commutative monad*.

Definition 2.7. Let $\mathcal{C}$ be a cartesian category. A *monad* (M, η) on $\mathcal{C}$ consists of:

- An assignment of an object MA to every object $A \in \mathcal{C}$.
- For every object $A \in \mathcal{A}$, a map $\eta_A : A \to MA$.
- For every map $f : A \to MB$, a *lift* $f^M : MA \to MA$.

This data is required to satisfy the following laws:

- (Unit) For any object A,
$$\eta_A^M = \mathrm{id}_{MA}.$$
- (Identity) For any map $f : A \to MB$,
$$f^M \circ \eta_A = f.$$
- (Composition) For any $f : A \to MB$ and $g : B \to MC$,
$$g^M \circ f^M = (g^M \circ f)^M.$$

From this data, we note that we can extend M into a functor $M : \mathcal{C} \to \mathcal{C}$ by sending $f : A \to B$ to $Mf \coloneqq (\eta_B \circ f)^M : MA \to MB$. Then $\eta : A \to MA$ is natural in A, and we get another natural transformation $\mu : MMA \to MA$ defined by lifting the identity: $\mu \coloneqq \mathrm{id}^M$. In fact, a monad may be equivalently defined as a functor $M : \mathcal{C} \to \mathcal{C}$ with natural transformations $\eta : A \to MA$ and $\mu : M^2A \to MA$ for which the following diagrams commutes:

$$\begin{array}{ccccc} MA & \xrightarrow{\eta} & M^2A & \xleftarrow{M\eta} & MA \\ & \searrow & \downarrow{\scriptstyle \mu} & \swarrow & \\ & & MA & & \end{array} \qquad \begin{array}{ccc} M^3A & \xrightarrow{\mu} & M^2A \\ {\scriptstyle M\mu}\downarrow & & \downarrow{\scriptstyle \mu} \\ M^2A & \xrightarrow[\mu]{} & MA \end{array} \tag{2.8}$$

For $f : A \to MB$, we can recover $f^M : MA \to MB$ from this definition of the monad M as $MA \xrightarrow{Mf} M^2B \xrightarrow{\mu} MB$.

A monad M is said to be *commutative* if there is a natural transformation $\sigma : MA \times MB \to M(A \times B)$ for which the following diagrams commute:

- $$\begin{array}{ccc} MA \times 1 & \xrightarrow{(MA\times\eta)\fatsemi\sigma_{A,1}} & M(A \times 1) \\ & \searrow^{\pi_1} & \downarrow{\scriptstyle M\pi_1} \\ & & MA \end{array} \tag{2.9}$$
- $$\begin{array}{ccc} 1 \times MA & \xrightarrow{(\eta\times MA)\fatsemi\sigma_{1,A}} & M(1 \times A) \\ & \searrow^{\pi_2} & \downarrow{\scriptstyle M\pi_2} \\ & & MA \end{array} \tag{2.10}$$
- $$\begin{array}{ccc} MA \times MB \times MC & \xrightarrow{MA\times\sigma_{B,C}} & MA \times M(B \times C) \\ {\scriptstyle \sigma_{A,B}\times MC}\downarrow & & \downarrow{\scriptstyle \sigma_{A,B\times C}} \\ M(A \times B) \times MC & \xrightarrow[\sigma_{A\times B,C}]{} & M(A \times B \times C) \end{array} \tag{2.11}$$
- $$\begin{array}{ccc} A \times B & \xrightarrow{\eta\times\eta} & MA \times MB \\ & \searrow^{\eta} & \downarrow{\scriptstyle \sigma} \\ & & M(A \times B) \end{array} \tag{2.12}$$
- $$\begin{array}{ccc} M^2A \times M^2B & \xrightarrow{\sigma\fatsemi M\sigma} & M^2(A \times B) \\ {\scriptstyle \mu\times\mu}\downarrow & & \downarrow{\scriptstyle \mu} \\ MA \times MB & \xrightarrow[\sigma]{} & M(A \times B) \end{array} \tag{2.13}$$

Remark 2.14. If you are familiar with the programming language Haskell, you will likely be familiar with the notion of monad. What we have called η_A here (which is

traditional in the category theory literature) is called `return` or `pure`. What we have called f^M for $f : A \to MB$ would, in haskell, be called `lift` and be defined by

```
lift = swap (>>=)
```

What we have called μ is called `join`. A monad in haskell is commutative if the following two programs have the same results:

```
f :: Monad m => m a -> m b -> m (a, b)
f ma ma = do
          a <- ma
          b <- mb
          return (a, b)

g :: Monad m => m a -> m b -> m (a, b)
g ma ma = do
          b <- mb
          a <- ma
          return (a, b)
```

That is, a monad is commutative when its order of execution doesn't matter.

Proposition 2.15. The powerset P is a commutative monad on the category of sets, with the following data:

- $\eta : A \to \mathrm{P}A$ sends $a \in A$ to the singleton set $\{a\}$.
- For $f : A \to \mathrm{P}B$, $f^{\mathrm{P}} : \mathrm{P}A \to \mathrm{P}B$ is defined by

$$f^{\mathrm{P}}(X) = \bigcup_{a \in X} f(a).$$

- $\sigma_{A,B} : \mathrm{P}A \times \mathrm{P}B \to \mathrm{P}(A \times B)$ is defind by

$$\sigma_{A,B}(X, Y) = \{(a, b) \mid a \in X, b \in Y\}.$$

Proof. We just need to check the laws.

- The function η^{P}_A takes a set $X \in \mathrm{P}A$ and yields $\bigcup_{x \in X} \{x\}$, which is equal to X.
- Let $f : A \to \mathrm{P}B$ be a function. Then $f^{\mathrm{P}}(\{a\}) = \bigcup_{a' \in \{a\}} f(a') = f(a)$ for any element $a \in A$.
- Let $f : A \to \mathrm{P}B$ and $g : B \to \mathrm{P}C$. For $X \in \mathrm{P}A$, we have

$$\begin{aligned} g^{\mathrm{P}} \circ f^{\mathrm{P}}(X) &= \bigcup_{b \in f^{\mathrm{P}}(X)} g(b) \\ &= \bigcup_{b \in \bigcup_{a \in X} f(a)} g(b) \end{aligned}$$

$$= \bigcup_{a \in X} \bigcup_{b \in f(a)} g(b)$$
$$= (g^{\mathrm{P}} \circ f)^{\mathrm{P}}.$$

It remains to show that the powerset monad is commutative. We note that P acts as a functor on $f : A \to B$ by

$$\mathrm{P}f(X) = (\eta_B \circ f)^{\mathrm{P}}(X) = \bigcup_{a \in X} \{f(a)\} = f[X].$$

sending a subset of A to its image in B. We also note that $\mu : \mathrm{P}^2 A \to \mathrm{P}A$ defined by $\mu = \mathrm{id}^{\mathrm{P}}_{\mathrm{P}A}$ sends a set S of subsets of S to its union $\bigcup s \in S s$.

- (Eq. (2.9)) Beginning with $(X, *) \in \mathrm{P}A \times 1$ (taking $1 \cong \{*\}$), we need to show that $\mathrm{P}\pi_1 \circ \sigma_{A,1}(X, \{*\}) = X$. Now, $\sigma_{A,1}(X, \{*\}) = \{(a, b) \mid a \in X, b \in \{*\}\}$; since there is just one $b \in \{*\}$, every $a \in X$ is paired with some b, so projecting out the first component gives us all of X.
- (Eq. (2.10)) This is the same as the above, but on the other side.
- (Eq. (2.11)) If we have $(X, Y, Z) \in \mathrm{P}A \times \mathrm{P}B \times \mathrm{P}C$, both sides of this diagram will give us $\{(a, b, c) \mid a \in X, b \in Y, c \in Z\}$.
- (Eq. (2.12)) For $(a, b) \in A \times B$, we have $\eta(a, b) = \{a, b\}$, and $\sigma(\eta(a), \eta(b)) = \{(x, y) \mid x \in \{a\},\ y \in \{b\}\}$.
- (Eq. (2.13)) Let S be a set of subsets of A and T a set of subsets of B. The bottom path gives us

  $$\sigma(\mu(S), \mu(T)) = \left\{ (x, y) \,\middle|\, x \in \bigcup_{s \in S} s,\ y \in \bigcup_{t \in T} t \right\}$$

  while taking the top path, we first get $\sigma(S, T) = \{(s, t) \mid s \in S,\ t \in T\}$ and then $M\sigma$ of that to get

  $$\sigma\left[\{(s, t) \mid s \in S,\ t \in T\}\right] = \{\{(x, y) \mid x \in s,\ y \in t\} \mid s \in S,\ t \in T\}.$$

  Finally, we take the union over this to get

  $$\mu(\mathrm{P}\sigma(\sigma(S, T))) = \bigcup_{s \in S,\, t \in T} \{(x, y) \mid x \in s,\ y \in t\}.$$

  These two paths are easily seen to give the same result.

□

Using the commutative monad structure of P, we can see that $U : \mathsf{PState_S} \times \mathsf{In_S} \to \mathsf{PState_S}$ is the composite

$$\mathsf{PState_S} \times \mathsf{In_S} \xrightarrow{\mathrm{id} \times \eta} \mathsf{PState_S} \times \mathrm{P}\mathsf{In_S} \xrightarrow{\sigma} \mathrm{P}(\mathsf{State_S} \times \mathsf{In_S}) \xrightarrow{\mathrm{update}^{\mathrm{P}}_{\mathsf{S}}} \mathsf{PState_S}.$$

This lets us iteratively apply the update function to a starting state or set of states.

It also lets us get the exposed variable out at the end. If we've been iteratively running a possibilistic system , then we won't know which state we are in but instead have a set $S \in \mathsf{PState}_\mathsf{S}$ of states we could possibly be in. Because of this, we can't directly apply $\mathsf{expose}_\mathsf{S} : \mathsf{State}_\mathsf{S} \to \mathsf{Out}_\mathsf{S}$, since it takes in a single state. But the monad structure of P gives us a function $\mathsf{Pexpose}_\mathsf{S} : \mathsf{PState}_\mathsf{S} \to \mathsf{POut}_\mathsf{S}$. Applying this to our current set of possible states gives us a set of possible outputs, which is the best we could hope to know.

Do Notation If we have a function $f : X \to Y$, we can think of this as mapping x in X to $f(x)$ in Y using "generalized elements" (see Remark 1.49). The *do notation* extends this way of writing morphisms in a cartesian category to include the action of a commutative monad M. The do notation is based on this simple equation for $f : X \to MY$:

$$\boxed{\begin{array}{l}\textbf{do}\\ \quad x \leftarrow m\\ \quad f(x)\end{array}} := f^M(m) \tag{2.16}$$

where m is an element of MX and $f : X \to MY$. For $M = \mathrm{D}$, we can understand the do notation in this way: m is a subset of X, $f^M(m)$ is the subset $\{f(x) \in Y \mid x \in m\}$. We see this reflected in the do notation; we can read it as saying "get an element x from m, and then apply $f(x)$ to it; join together all the results." As we see more monads, we will see that a similar story can be told about them using the do notation.

There are a few rules for do notation which correspond to the laws for a monad. We can discover them by using Eq. (2.16) to expand out a few terms. First of all, since $\eta^M = \mathrm{id}_{MX}$, if m is an element of MX, then

$$\boxed{\begin{array}{l}\textbf{do}\\ \quad x \leftarrow m\\ \quad \eta(x)\end{array}} = m$$

Next, since $\eta \fatsemi f^M = f$, we find that

$$\boxed{\begin{array}{l}\textbf{do}\\ \quad x' \leftarrow \eta(x)\\ \quad f(x')\end{array}} = f(x)$$

Finally, since $f^M \fatsemi g^M = (f \fatsemi g^M)^M$, we find that

$$\boxed{\begin{array}{l}\textbf{do}\\ \quad y \leftarrow \boxed{\begin{array}{l}\textbf{do}\\ \quad x \leftarrow m\\ \quad f(x)\end{array}}\\ \quad g(y)\end{array}} = \boxed{\begin{array}{l}\textbf{do}\\ \quad x \leftarrow m\\ \quad \boxed{\begin{array}{l}\textbf{do}\\ \quad y \leftarrow f(x)\\ \quad g(y)\end{array}}\end{array}}$$

Because these two expressions with nested do's are equal, we can simplify our notation by writing them as:

$$\boxed{\begin{array}{l}\textbf{do}\\ \quad x \leftarrow m\\ \quad y \leftarrow f(x)\\ \quad g(y)\end{array}}$$

So far, we haven't used any pairs (x, y) in our do notation. To use pairs, we need our monad to be commutative. We can write down two expressions, assuming m_1 is an element of MX and m_2 is an element of MY. A monad is commutative precisely when these two expressions are equal:

$$\boxed{\begin{array}{l}\textbf{do}\\ \quad x \leftarrow m_1\\ \quad y \leftarrow m_2\\ \quad \eta(x, y)\end{array}} = \boxed{\begin{array}{l}\textbf{do}\\ \quad y \leftarrow m_2\\ \quad x \leftarrow m_1\\ \quad \eta(x, y)\end{array}}$$

When they are both equal, they are $\sigma(m_1, m_2)$, where $\sigma : MX \times MY \to M(X \times Y)$ is from the definition of a commutative monad. This lets us describe morphisms quite nicely. For example, given $f : X \to MY$, $g : Z \to MW$, and $h : Y \times W \to MQ$, we may define

$$\boxed{\begin{array}{l}\textbf{do}\\ \quad y \leftarrow f(x)\\ \quad w \leftarrow g(z)\\ \quad h(y, w)\end{array}}$$

which desugars to the composite

$$X \times Z \xrightarrow{f \times g} MY \times MW \xrightarrow{\sigma} M(Y \times W) \xrightarrow{h^M} MQ.$$

In particular, to iterate a system S with update $\mathrm{update}_{\mathsf{S}} : \mathsf{State}_{\mathsf{S}} \times \mathsf{In}_{\mathsf{S}} \to \mathsf{DState}_{\mathsf{S}}$, we can define

$$U(S, i) := \boxed{\begin{array}{l}\textbf{do}\\ \quad s \leftarrow S\\ \quad \mathrm{update}_{\mathsf{S}}(s, i)\end{array}}$$

2.2 Stochastic systems

Possibility is not the only kind of non-determinism. When studying how things change in the world, we often notice that we can only predict how likely some change will be, and not precisely which change will occur. If instead of asking whether a change is possible, we ask how *probable* it is, we arrive at a notion of probabilistic or *stochastic* system.

The notion of a stochastic system is based on the idea that there should be a probability of a given change occuring, conditioned upon the current state. A useful way

to formulate the notion of conditional probability is the notion of *stochastic map*. A stochastic map from A to B is a function which takes an $a \in A$ and yields a probability distribution $p(- \mid a)$ on elements of B which we think of as likelyhoods conditioned on a. We can make this more precise using the notion of monad.

Definition 2.17. For a set A, the set $\mathrm{D}A$ is the set of finitely supported probability distributions on A. A probability distribution on A is a function $p : A \to [0,1]$ which takes non-zero values at only finitely many elements of A, and for which

$$\sum_{a \in A} p(a) = 1.$$

This sum makes sense because only finitely many elements of A give non-zero $p(a)$.

The elements of $\mathrm{D}A$ can be identified with *formal convex combinations* of elements of A. A formal convex combination

$$\sum_{a \in X} \lambda_a a$$

of elements of A consists of a finite and inhabited[a] subset $X \subseteq A$ of elements together with a function $\lambda_{(-)} : X \to (0,1]$ assigning each $a \in X$ a coefficient λ_a such that $\sum_{a \in X} \lambda_a = 1$.

$$\mathrm{D}A = \left\{ \sum_{a \in X} \lambda_a a \;\middle|\; X \subseteq A,\ X \text{ finite and inhabited},\ \lambda_{(-)} : X \to (0,1],\ \sum_{a \in X} \lambda_a = 1 \right\}.$$

[a] That is, there is some $a \in X$.

Example 2.18. Let's see what $\mathrm{D}A$ looks like for a few different sets A:

1. If $A = \{a\}$ has a single element, then there is only one inhabited subset $X \subseteq A$ (namely $X = A$) and since the coefficients of any convex linear combination must sum to 1, the coefficient of the single element must be 1. So $\mathrm{D}\{a\} = \{1 \cdot a\}$ contains a single element.
2. If $A = \{a, b\}$, things get more interesting. Now there are three possible subsets X: $\{a\}$, $\{b\}$, and $\{a, b\}$. A convex combination with a single element must have coefficient 1, so we at least have the convex combinations $1 \cdot a$ and $1 \cdot b$. But for the set $\{a, b\}$, we have the convex combination $\lambda_a a + \lambda_b b$ where $\lambda_a + \lambda_b = 1$ and $\lambda_a, \lambda_b > 0$. If we make the association of $1 \cdot a$ with $1 \cdot a + 0 \cdot b$, and similarly for $1 \cdot b$, then we can see that

   $$\mathrm{D}\{a, b\} = \{\lambda a + (1 - \lambda) b \mid \lambda \in [0,1]\}$$

   which is bijective with the closed interval $[0,1]$.
3. In general, if A is a finite set with n elements, then $\mathrm{D}A$ can be identified with the *standard n-simplex*, that is, the set of solutions to the equation $\sum_{i=1}^{n} \lambda_i = 1$ for

$\lambda_i \in [0,1]$.

$$\mathsf{Dn} \cong \{(\lambda_1, \ldots, \lambda_n) \in [0,1]^n \mid \sum_{i=1}^{n} \lambda_i = 1\}.$$

Definition 2.19. A *stochastic map* from a set A to a set B is a function $f : A \to \mathrm{D}B$, assigning each $a \in A$ to a probability distribution $f(a)$ on B.

If the sets A and B are finite, then we can write a stochastic map $f : A \to \mathrm{D}B$ as a *stochastic matrix*. This is an $B \times A$ matrix whose ba-entry is $f(a)(b)$. Any matrix of positive entries where every column sums to 1 arises as the stochastic matrix of a stochastic map.

We think of a stochastic map $f : A \to \mathrm{D}B$ as giving a bunch of conditional probabilities

$$p(b \mid a) \coloneqq f(a)(b).$$

Example 2.20. If I see someone enter the office soaking wet, it is likely to have been raining. If they are dry, it may be less likely that it was raining; but, if they have an umbrella, then they might be dry but it is still more likely that it was raining. We can express these various conditional probabilities as a stochastic function

$$\{\texttt{wet}, \texttt{dry}\} \times \{\texttt{umbrella}, \texttt{no}-\texttt{umbrella}\} \to \mathrm{D}\{\texttt{raining}, \texttt{not}-\texttt{raining}\}.$$

We can describe this stochastic function in full by giving its stochastic matrix:

$$\begin{array}{c} \\ \texttt{raining} \\ \texttt{not-raining} \end{array} \begin{array}{c} \begin{array}{cccc} \texttt{(wet,umbrella)} & \texttt{(wet,no-umbrella)} & \texttt{(dry,umbrella)} & \texttt{(dry,no-umbrella)} \end{array} \\ \begin{bmatrix} .9 & .9 & .5 & .3 \\ .1 & .1 & .5 & .7 \end{bmatrix} \end{array}$$

A stochastic system is a system whose dynamics is given by a stochastic map.

Definition 2.21. A *stochastic system* S, also written as

$$\begin{pmatrix} \mathsf{update}_\mathsf{S} \\ \mathsf{expose}_\mathsf{S} \end{pmatrix} : \begin{pmatrix} \mathsf{State}_\mathsf{S} \\ \mathsf{State}_\mathsf{S} \end{pmatrix} \leftrightarrows \begin{pmatrix} \mathsf{In}_\mathsf{S} \\ \mathsf{Out}_\mathsf{S} \end{pmatrix},$$

consists of:

- a set $\mathsf{State}_\mathsf{S}$ of *states*;
- a set Out_S of *values for exposed variables*, or *outputs* for short;
- a set In_S of *parameter values*, or *inputs* for short;
- a function $\mathsf{expose}_\mathsf{S} : \mathsf{State}_\mathsf{S} \to \mathsf{Out}_\mathsf{S}$, the *exposed variable of state* or *expose* function, which takes a state to the output it yields; and

- a function $\text{update}_S : \text{State}_S \times \text{In}_S \to \text{PState}_S$, where DState_S is the set of subsets of State_S. This is the *dynamics* or *update* function which takes a state and a parameter and gives the set of possible next states.

Remark 2.22. A stochastic system is often called a *Markov process*.

Example 2.23. A simple, but entertaining example of a stochastic system is a text generator. Suppose we have a big pile of text — say, the speeches of a famous person — and we want to generate some text that looks like it was written by the same author. There are many sophisticated ways to do this, but here's a very bone-headed approach. We will look at the text in 5-character length sequences, and ask: how likely is for a given character to follow this 5-character sequence.

For example, if our text is

To be or not to be, that is the question.

Then we can see that there is a 50% chance that " " and a 50% chance that "," follows the 5-character sequence "to be". Of course, such a small sample woulnd't give us very useful statistics, but if we use the combined works of Shakespeare, we might get a better sense of what is likely to occur next.

Now we build a stochastic system S which will generate text. We take State_S to be length 5 sequences of characters from our alphabet Alphabet: $\text{State}_S = \text{Alphabet}^5$. We will expose the first character in the sequence: $\text{Out}_S = \text{Alphabet}$ and $\text{expose}_S(s) = s_1$. We don't need any input to the system: $\text{In}_S = \{*\}$. Now, $\text{update}_S(s)$ will assign to a sequence (s_2, s_3, s_4, s_5, c) the probability that the character c follows the sequence $s = (s_1, s_2, s_3, s_4, s_5)$ in our sample text, and assign all other sequences the probability 0.

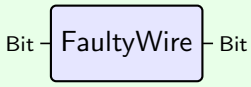
If we run our stochastic text generator over time, it will produce a stream of characters that have the statistical properties of our sample text. As simple minded as this approach is, it can produce some fun results:

> HAMLET
>
> Whose image even but now appear'd again!
>
> HORATIO
>
> From top to toe?
>
> FRANCISCO
>
> Bernardo, on their promise, as it is a course to any moment leisure, but to persever Than the cock, that this believe Those friend on Denmark Do not dull thy name with a defeated him yesternight.

Example 2.24. A *stochastic source process* is a stochastic system S with no inputs $\mathsf{In}_S = 1$. Such a stochastic system would be boxed up like this:

These are means by which random streams of outputs can be generated. In Example 2.23, we described a stochastic source process that produced Shakespearean writing (of a stunted sort). In his seminal paper "A mathematical theory of communication", Claude Shannon imagined communicators as stochastic source processes sending somewhat random language through various communication channels. This point of view is still used today to model communications that have some complicated structure which, not knowing how that structure is generated in particular, are best modeled as somewhat random processes.

Example 2.25. We can model a faulty wire as a stochastic system of the following sort:



We will define FaultyWire as follows:

- A faulty wire will either have `good` contact, `partial` contact, or `missing` contact, and it will be carrying a `high` or `low` charge:

$$\mathsf{State}_{\mathsf{FaultyWire}} := \{\texttt{high}, \texttt{low}\} \times \{\texttt{good}, \texttt{partial}, \texttt{missing}\}.$$

- The faulty wire will take in either a `high` or `low`:

$$\mathsf{In}_{\mathsf{FaultyWire}} = \mathsf{Out}_{\mathsf{FaultyWire}} = \mathsf{Bit} = \{\texttt{high}, \texttt{low}\}.$$

- The faulty wire exposes its current charge:

$$\mathsf{expose}_{\mathsf{FaultyWire}}(b, s) = b.$$

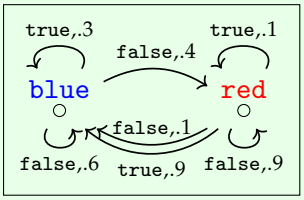
- The faulty wire will try and set its charge to the charge on the incoming wire, but if it is has bad contact, this won't succeed and it will have low charge. It's contact also has a small chance to decay.

$$\begin{aligned}
&\mathsf{update}_{\mathsf{FaultyWire}}((b, \texttt{good}), i) = .99(i, \texttt{good}) + .01(i, \texttt{partial}),\\
&\mathsf{update}_{\mathsf{FaultyWire}}((b, \texttt{partial}), i) = .50(i, \texttt{partial}) + .49(\texttt{low}, \texttt{partial}) + .01(\texttt{low}, \texttt{missing}),\\
&\mathsf{update}_{\mathsf{FaultyWire}}((b, \texttt{missing}), i) = (\texttt{low}, \texttt{no}).
\end{aligned}$$

When wiring up our systems, if we put a faulty wire in between, we will introduce the probability of the failure of this wire to communicate into the model.

Example 2.26. We can draw transition diagrams for stochastic systems, just like we do for deterministic and possibilistic systems. This time, we will label each transition with the probability that it occurs. We just have to make sure that the probability labels on all the outgoing transitions with the same input label on any state sum to 1.

For example, here is a stochastic system drawn as a transition diagram:



(2.27)

The set D of probability distributions is a commutative monad, like the powerset P monad.

Proposition 2.28. The assignment of a set A to its set $\mathrm{D}A$ of probability distributions is a commutative monad with the data:

- $\eta_A : A \to \mathrm{D}A$ sends every element a to its *Dirac delta distribution* $\eta_A(a) = 1 \cdot a$ which assigns probability 1 to a and probability 0 to everything else. As a convex linear combination, it looks like this:
$$\eta_A(a) = \sum_{a' \in \{a\}} 1 \cdot a'$$
- Given a stochastic map $f : A \to \mathrm{D}B$ sending $a \in A$ to $f(a) = \sum_{b \in Y_a} \rho_{ba} b$, we can *push forward* a probability distribution $p = \sum_{a \in X} \lambda_a a$ on A to a probability distribution
$$f^{\mathrm{D}}(p) \coloneqq \sum_{b \in \bigcup_{a \in X} Y_a} \left(\sum_{a \in X} \rho_{ba} \lambda_a \right) b = \sum_{a \in X} \sum_{b \in Y_a} \rho_{ba} \lambda_a b$$
on B. In classical terms, this says that given conditional probabilities $p(b \mid a) \coloneqq f(a)(b)$ and any prior distribution $p(a) \coloneqq \lambda_a$, we can form a posterior distribution $p(b) \coloneqq \sum_{a \in A} p(b \mid a) p(a)$.
- Given a probability distribution $\sum_{a \in X} \lambda_a a$ on A and $\sum_{b \in Y} \mu_b b$ on B, we can form their joint distribution
$$\sum_{(a,b) \in X \times Y} \lambda_a \mu_b (a, b)$$
on $A \times B$. This gives us $\sigma : \mathrm{D}A \times \mathrm{D}B \to \mathrm{D}(A \times B)$. In classical terms, this says that the probability of two independent events is the product of their probabilities: $p(a, b) = p(a)p(b)$.

Proof. We check the laws:

- If we push forward a distribution $p = \sum_{a \in X} \lambda_a a$ along $\eta_A : A \to \mathrm{D}A$, we get
$$\eta_A^{\mathrm{D}}(p) = \sum_{a \in X} \sum_{a' \in \{a\}} 1 \cdot \lambda_{a'} a' = \sum_{a \in X} \lambda_a a.$$
- For a stochastic map $f : A \to \mathrm{D}B$, we aim to show that pushing forward the Dirac delta distribution $\eta_A(a)$ along f gives $f(a) = \sum_{b \in Y_a} \lambda_{ba} b$. The definition of push forward gives us
$$f^{\mathrm{D}}(\eta_A(a)) = \sum_{a' \in \{a\}} \sum_{b \in Y_{a'}} \lambda_{ba} \cdot 1 \cdot b = \sum_{b \in Y_a} \lambda_{ba} b.$$
- Given stochastic functions $f : A \to \mathrm{D}B$ and $g : B \to \mathrm{D}C$, we need to show that $g^{\mathrm{D}}(f^{\mathrm{D}}(p)) = (g^{\mathrm{D}} \circ f)^{\mathrm{D}}(p)$. Let
$$p = \sum_{a \in X} \lambda_a,$$

$$f(a) = \sum_{b \in Y_a} \rho_{ba} b,$$
$$g(b) = \sum_{c \in Z_b} \gamma_{cb} c.$$

Then we see that

$$g^{\mathrm{D}}(f(a)) = \sum_{c \in \bigcup_{b \in \bigcup_{a \in X} Y_a}} \gamma_{cb} \rho_{ba} c$$

so that, finally

$$\begin{aligned} g^{\mathrm{D}}(f^{\mathrm{D}}(p)) &= g^{\mathrm{D}}\left(\sum_{a \in X} \sum_{b \in Y_a} \rho_{ba} \lambda_a c \right) \\ &= \sum_{a \in X} \sum_{b \in Y_a} \sum_{c \in Z_b} \gamma_{cb} \rho_{ba} \lambda_a c \\ &= (g^{\mathrm{D}} \circ f)^{\mathrm{D}}(p). \end{aligned}$$

Next, we check that the laws of a commutative monad hold. We note that for a function $f : A \to B$, the function $\mathrm{D}f = (\eta_B \circ f)^{\mathrm{D}}$ is defined by

$$\mathrm{D}f\left(\sum_{a \in X} \lambda_a a \right) = \sum_{a \in X} \sum_{b \in \{f(a)\}} \lambda_a b = \sum_{a \in X} \lambda_a f(a).$$

Furthermore, $\mu : \mathrm{D}^2 A \to \mathrm{D}A$ sends a formal convex combination $\sum_i \lambda_i p_i$ of probability distributions to the *actual* convex combination of those probability distributions, namely the distribution

$$\mu\left(\sum_i \lambda_i p_i \right)(a) \coloneqq \sum_i \lambda_i p_i(a).$$

- (Eq. (2.9)) The unit on $1 \cong \{*\}$ sends $*$ to the distribution $1 \cdot *$. So, $\sigma(p, 1) = \sum_{(a,*) \in X \times 1} \lambda_a \cdot 1 \cdot (a, *)$, and projecting out again gives us $p = \sum_{a \in X} \lambda_a a$.
- (Eq. (2.10)) The same, but on the other side.
- (Eq. (2.11)) Suppose that we have

$$p = \sum_{a \in X} p_a a,$$
$$q = \sum_{b \in Y} q_b b,$$
$$r = \sum_{c \in Z} r_c c.$$

The both paths of Eq. (2.11) give us the distribution

$$\sum_{(a,b,c) \in X \times Y \times Z} p_a q_b r_c (a, b, c).$$

- (Eq. (2.12)) This is asking whether $\delta_{(a,b)} = \delta_a \delta_b$ as distributions on $A \times B$, which they are.
- (Eq. (2.13)) Let $\sum_i \lambda_i p_i$ be an element of $\mathrm{D}\mathrm{D}A$, and similarly let $\sum_j \rho_j q_j$ be an element of $\mathrm{D}\mathrm{D}B$. Following the bottom path around, we get

$$\sigma\left(\mu\left(\sum_i \lambda_i p_i\right), \mu\left(\sum_j \rho_j q_j\right)\right)(a,b) = \left(\sum_i \lambda_i p_i(a)\right)\left(\sum_j \rho_j q_j(b)\right) = \sum_i \sum_j \lambda_i \rho_j p_i(a) q_j(b).$$

Meanwhile,

$$\sigma\left(\sum_i \lambda_i p_i, \sum_j \rho_j q_j\right) = \sum_i \sum_j \lambda_i \rho_j (p_i, q_j).$$

and taking $\mathrm{D}\sigma$ of that gives

$$\sum_i \sum_j \lambda_i \rho_j p_i q_j$$

which means that finally

$$\mu\left(\mathrm{D}\sigma\left(\sigma\left(\sum_i \lambda_i p_i, \sum_j \rho_j q_j\right)\right)\right)(a,b) = \sum_i \sum_j \rho_j p_i(a) q_j(b).$$

□

Exercise 2.29. Let $f : \mathsf{n} \to \mathrm{D}\mathsf{m}$ and $g : \mathsf{m} \to \mathrm{D}\mathsf{k}$ be stochastic maps. Note that we can interpret f as an $m \times n$ stochastic matrix F, and similarly g as a $k \times m$ stochastic matrix G. Show that the stochastic map $g^{\mathrm{D}} \circ f$ is associated to the stochastic matrix GF. ◊

Just as the commutative monad structure of P helped us iterate possibilistic systems and get the set of possible output values from them, so the commutative monad structure of D helps us iterate stochastic systems and get a probability distribution of likely output values from them.

Given a stochastic system S, we have $\mathrm{update}_{\mathsf{S}} : \mathrm{State}_{\mathsf{S}} \times \mathrm{In}_{\mathsf{S}} \to \mathrm{D}\mathrm{State}_{\mathsf{S}}$. From this, we can get a stochastic map:

$$\mathrm{D}\mathrm{State}_{\mathsf{S}} \times \mathrm{In}_{\mathsf{S}} \xrightarrow{\mathrm{id}\times\eta} \mathrm{D}\mathrm{State}_{\mathsf{S}} \times \mathrm{D}\mathrm{In}_{\mathsf{S}} \xrightarrow{\sigma} \mathrm{D}(\mathrm{State}_{\mathsf{S}} \times \mathrm{In}_{\mathsf{S}}) \xrightarrow{\mathrm{update}_{\mathsf{S}}^{\mathrm{D}}} \mathrm{D}\mathrm{State}_{\mathsf{S}}$$

which will let us iterate. We can see that this sends a probability distribution p on states and an input i to the distribution

$$s \mapsto \sum_{s' \in \mathrm{State}_{\mathsf{S}}} p(s') \mathrm{update}_{\mathsf{S}}(s', i)(s).$$

2.3 Monadic doctrines and the Kleisli category

We have now seen two sorts of non-determinism expressed by commutative monads. To each of these we associated a doctrine:

- To the powerset monad P, we associated the doctrine of possibilistic systems. This is because a map $f : A \to \mathrm{P}B$ is a *possibilistic map* — it assigns a set of *possible images* to each element $a \in A$.
- To the probability distribution monad D, we associated the doctrine of stochastic system. This is because a map $f : A \to \mathrm{D}B$ is a stochastic map.

In general, for any commutative monad M we call a map of the form $f : A \to MB$ a *Kleisli map*. The structure of a monad on M lets us compose Kleisli maps, giving us the *Kleisli category* of the monad. The commutativity then makes the Kleisli category into a symmetric monoidal category.

Definition 2.30. Let $M : \mathcal{C} \to \mathcal{C}$ be a commutative monad on a cartesian category. The *Kleisli category* $\mathbf{Kl}(M)$ is defined as follows:

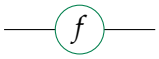
- The objects of $\mathbf{Kl}(M)$ are the same as those of $\mathcal{C}$.
- A map $f : A \rightsquigarrow B$ in $\mathbf{Kl}(M)$ is a map $f : A \to MB$ in $\mathcal{C}$.
- The identity $\mathrm{id}_A : A \rightsquigarrow A$ is $\eta_A : A \to MA$.
- For $f : A \rightsquigarrow B$ and $g : B \rightsquigarrow C$, their composite is $f \fatsemi g^M : A \to MC$. In do notation, the Kleisli composite is given by

$$(f \fatsemi g)(a) := \boxed{\begin{array}{l}\textbf{do} \\ \quad b \leftarrow f(a) \\ \quad g(b)\end{array}}.$$

Since $g^M = Mg \fatsemi \mu$, the Kleisli composite may be equivalently defined as $f \fatsemi Mg \fatsemi \mu$. The Kleisli category of M becomes a symmetric monoidal structure with with the tensor $A \times B$ and 1. Note that although $A \times B$ is cartesian in $\mathcal{C}$, it will rarely be cartesian in $\mathbf{Kl}(M)$.

We can understand Kleisli composition a bit better if we introduce a graphical language for monads.[1] This will also help us later in **??** when we learn about *biKleisli* composition. We will draw an object of our category $X \in \mathcal{C}$ as a string:

and a map $f : X \to Y$ as a bead:



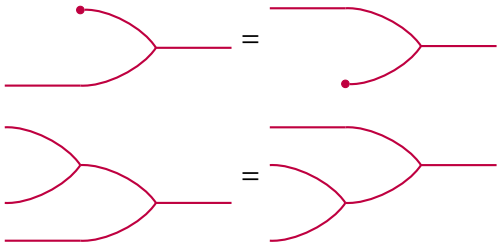
Composition is drawn by connecting strings, and the identity map on X is represented by the same string which represents X. We will draw our monad $M : \mathcal{C} \to \mathcal{C}$ as a red

[1] If you know of it, this is just the usual string diagram language for 2-categories.

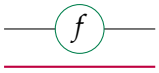
string:

We can draw the natural transformations $\eta : \mathrm{id}_{\mathcal{C}} \Rightarrow M$ and $\mu : M^2 \Rightarrow M$ as

respectively. The laws Eq. (2.8) can be written as:

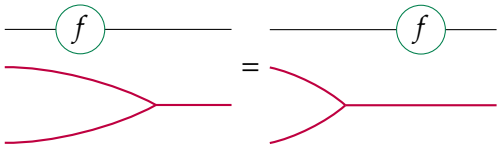


The map $Mf : MX \to MY$ on objects is written:

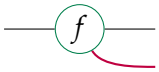


Note that functoriality is baked in to this string diagram notation; the following diagram could either be interpreted as $Mf \fatsemi Mg$ or $M(f \fatsemi g)$, which are equal by the functoriality of M:

The naturality of η and μ is also baked into this notation; it just means we can move them independently of the beads representing functions:

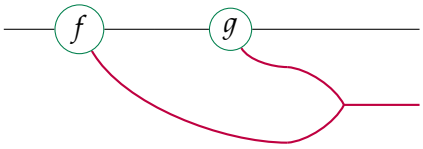


With these conventions in hand, we can now represent a Kleisli map $f : X \to MY$ as

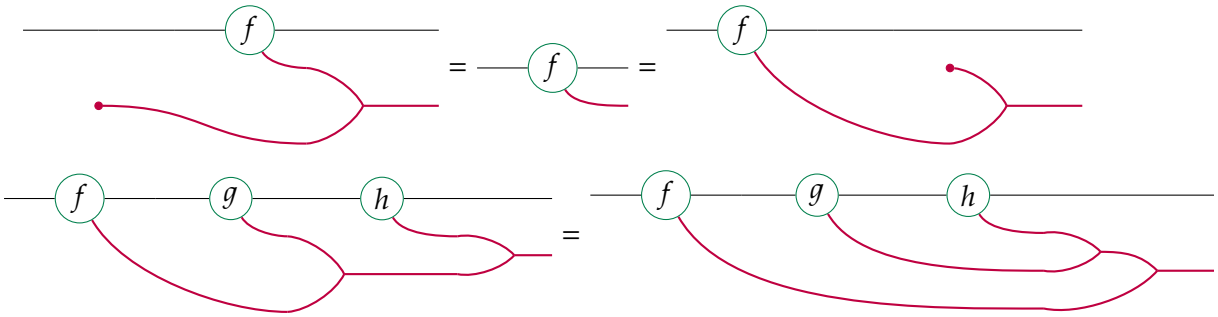


The unit $\eta : X \to MX$ is written

The composition of Kleisli maps $f : X \to MY$ and $g : Y \to MZ$ is then given by



We can use these string diagrams to easily check that **Kl**(M) is actually a category. We use the monad laws Eq. (2.8):



Example 2.31. The Kleisli category **Kl**(P) of the powerset monad P is the category of *multi-valued maps*. A Kleisli map $f : A \to PB$ assigns to each $a \in A$ a subset $f(a) \subseteq B$ of possible images of a. Given another Kleisli map $g : B \to PC$, their composite in the Kleisli category $g^{\mathrm{P}} \circ f : A \to PC$ sends $a \in A$ to the union $\bigcup_{b \in f(a)} g(b)$. In other words, a possible image of $g \circ f$ is any possible image of g of any possible image of f.

Example 2.32. The Kleisli category **Kl**(D) of the probability monad D is the category of *stochastic maps*. A Kleisli map $f : A \to DB$ assigns to each $a \in A$ a probability distribution $f(a)$ on B. Given another Kleisli map $g : B \to DC$, their composite $g^{\mathrm{D}} \circ f : A \to DC$ in the Kleisli category sends a to the probability distribution $c \mapsto \sum_{b \in B} f(a)(b) \cdot g(b)(c)$. That is, since c is the image of a under $g \circ f$ if there is a b which is the image of a under f and c is the image of b under g, the probability that c is the image of a is the probability of their being such a b.

Thinking of stochastic maps as conditional probabilities, where $f : A \to DB$ expresses the conditional probability $p(b \mid a) = f(a)(b)$, then we see that $p(c \mid a) = \sum_{b \in B} p(b \mid a)p(c \mid b)$ as we expect from conditional probabilities.

Now we encompass all our non-deterministic examples in a single definition.

Definition 2.33. Let $M : \mathcal{C} \to \mathcal{C}$ be a commutative monad. A (discrete-time) M-system S, also written as

$$\begin{pmatrix} \mathsf{update}_{\mathsf{S}} \\ \mathsf{expose}_{\mathsf{S}} \end{pmatrix} : \begin{pmatrix} \mathsf{State}_{\mathsf{S}} \\ \mathsf{State}_{\mathsf{S}} \end{pmatrix} \leftrightarrows \begin{pmatrix} \mathsf{In}_{\mathsf{S}} \\ \mathsf{Out}_{\mathsf{S}} \end{pmatrix},$$

is a system whose dynamics is given by a Kleisli map for M. It consists of:

- an object $\mathsf{State}_{\mathsf{S}}$ of *states*;
- an object $\mathsf{Out}_{\mathsf{S}}$ of *values for exposed variables*, or *outputs* for short;
- an object In_{S} of *parameter values*, or *inputs* for short;
- a map $\mathsf{expose}_{\mathsf{S}} : \mathsf{State}_{\mathsf{S}} \to \mathsf{Out}_{\mathsf{S}}$, the *exposed variable of state* or *expose* map, which

takes a state to the output it yields; and
- a Kleisli map $\text{update}_S : \text{State}_S \times \text{In}_S \to M\text{State}_S$. This is the *dynamics* or *update* map which takes a state and a parameter and gives the next state in a non-deterministic way determined by M.

This will let us more swiftly describe new non-deterministic doctrines.

For example, suppose that our system is free to choose which state it transitions to next, but there's a catch. For any state s and input parameter i, there will be a cost $\text{update}_S(s, i)(s') \in [0, \infty]$ associated to each other state s' — the cost of transitioning from s to s' given the parameter i. A cost of 0 means that this transition is free; a cost of ∞ means it is prohibitively expensive, or impossible.

Definition 2.34. We will define a monad Cost on the category of sets. We think of a Kleisli map $f : A \to \text{Cost}(B)$ as assiging the best-case cost of producing a $b \in B$ from a given $a \in A$. For practical reasons, we assume that only finitely many $b \in B$ are possible (that is, have finite cost) to produce from an $a \in A$.

- For a set A,
$$\text{Cost}(A) := \{c : A \to [0, \infty] \mid \{a \in A \mid c(a) < \infty\} \text{ is finite}\}$$
is the set of cost functions $c : A \to [0, \infty]$ which assign finite values to only finitely many elements of A.
- For a set A, $\eta^{\text{Cost}} : A \to \text{Cost}(A)$ assumes that we can only produce what we have, but that if we already have it, it's free. Formally:
$$\eta^{\text{Cost}}(a)(a') := \begin{cases} 0 & \text{if } a = a' \\ \infty & \text{otherwise} \end{cases}$$
- For a map with cost $f : A \to \text{Cost}(B)$, we define $f^{\text{Cost}} : \text{Cost}(A) \to \text{Cost}(B)$ by
$$f^{\text{Cost}}(c)(b) := \min_{a \in A} c(a) + f(a)(b).$$
That is, given costs on elements of A and conditional costs on elements of B given by f, the cost of an element of B is the cost of getting an $a \in A$ together with the cost of producing b from that a. So, the best case cost of such a b is the minimum over all $a \in A$ of the total cost of producing b from a. We note that the minimum is achieved because only finitely many of the costs are finite.
- Given sets A and B, the cost of having an element of A and an element of B is the sum of their costs.
$$\sigma(c, c')(a, b) := c(a) + c'(b).$$

Remark 2.35. We will prove that Definition 2.34 does indeed give a commutative monad in the upcoming **??**.

Now we can quickly define our new sort of non-determinism.

Definition 2.36. A (discrete-time) system with costs is a Cost-system.

Example 2.37. Suppose we are trying to complete a project Proj that involves a number of steps. Let Steps be the set of steps involved. The state of our project at any given time is the set of steps we have completed so far: $\mathsf{State}_{\mathsf{Proj}} := \mathsf{PSteps}$. Now, we may not want to show everyone exactly how our project is going, just that it has hit certain milestones. So we can let $\mathsf{Out}_{\mathsf{Proj}} := \mathsf{Milestones}$ be our set of milestones and $\mathsf{expose}_{\mathsf{Proj}} : \mathsf{State}_{\mathsf{Proj}} \to \mathsf{Out}_{\mathsf{Proj}}$ send each project state to the most recent milestone completed.

Now, in any project, there are some external conditions to be dealt with. Let $\mathsf{In}_{\mathsf{Proj}} = \mathsf{Externalities}$ be the set of these externalities. We can assume that there is a cost associated to choosing a next step to take which depends not only on what steps have been completed so far but also on the current external conditions: that is, we can assume we have a function $\mathsf{cost} : \mathsf{State}_{\mathsf{Proj}} \times \mathsf{In}_{\mathsf{Proj}} \to \mathsf{Cost}(\mathsf{Steps})$, and that $\mathsf{cost}(s, i)(x) = 0$ whenever $x \in s$ is a step we have already completed.[a] Given this, we can define the update of our project system as

$$\mathrm{update}_{\mathsf{Proj}}(s, i)(s') := \sum_{x \in s'} \mathrm{cost}(s, i)(x).$$

This tells us that the cost moving from having completed the steps s to having completed the steps s' given external conditions i is the sum of the cost of completing each step in s' which is not in s.

The crucial question we want to ask of this model is: how much will the project cost in the best case scenario, given a sequence of external conditions? That is, we will iterate the action of the system through the sequence of paramters starting at $\emptyset \in \mathsf{State}_{\mathsf{Proj}}$, and then ask the cost of $\mathsf{Steps} \in \mathsf{State}_{\mathsf{Proj}}$ at the end.

[a] Although one could imagine this instead as a "maintenance" cost of maintaining the completion of that step.

We took Cost to be the monad of best case costs. Let's show that there is also a monad $\mathrm{Cost}^{\max}$ of worst case costs. Everything will be the same, but instead of

$$f^{\mathrm{Cost}}(c)(b) := \min_{a \in A} c(a) + f(a)(b),$$

we will have

$$f^{\mathrm{Cost}^{\max}}(c)(b) := \max_{a \in A} c(a) + f(a)(b).$$

It is worth noting that this formula has a formal similarity to the following formula:

$$f^{R}(c)(b) := \sum_{a \in A} c(a) \cdot f(a)(b).$$

which resembles matrix multiplication. This is indeed the case; for any sort of (commutative) scalars, we get a monad that reproduces matrix arithmetic with those scalars. An appropriate set of scalars is called a *commutative rig*.

Definition 2.38. A *commutative rig* (for "ri**n**g without **n**egatives"[a]) is a set R equipped with an abelian group structure $(R, +, 0)$ and a commutative monoid structure $(R, \cdot, 1)$ such that

$$a \cdot (b + c) = a \cdot b + a \cdot c$$

for all a, b, $c \in R$.

[a] Rigs are also sometimes referred to as "semirings".

Example 2.39. The following are important examples of rigs:

1. The natural numbers $\mathbb{N}$ with their usual addition and multiplication form a rig. Similarly, the non-negative rationals and reals form rigs with their usual addition and multiplication.
2. Any ring is a rig. In particular, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ are all rigs with their usual addition and multiplication.
3. The *tropical rigs* are rigs where "addition" is actually minimum or maximum, and "multiplication" is actually addition. In particular, the *rig of best-case costs* $[0, \infty]$ is a rig with min as its addition and + as its multiplication. In this rig, distributivity looks like

   $$a + \min\{b, c\} = \min\{a + b, a + c\},$$

   and a linear combination looks like

   $$\min_{i \in I} c_i + x_i.$$

   The additive unit is ∞, and the multiplicative unit is 0.
   Similarly, there is a *rig of worst-case costs* on $[0, \infty]$ with max as addition and + as multiplication. This rig is remarkable in that its additive and multiplicative unit are the same; they are both 0.
4. In fact, any ordered commutative monoid $(M, +, 0, \leq)$ (where if $a \leq b$, then $c + a \leq c + b$) which admits joins $a \vee b$ (that is, least upper bounds) can be made into a commutative rig with addition given by $\vee$ and multiplication given by +.

Proposition 2.40. For any commutative rig R, there is a commutative monad $R \otimes - : \mathbf{Set} \to \mathbf{Set}$ defined by

- $R \otimes X$ is the set of R-linear combinations of elements of X.
- $\eta : X \to R \otimes X$ sends x to the linear combination $\cdot x$.
- For $f : X \to R \otimes Y$, we have $f^R : R \otimes X \to R \otimes Y$ defined by

$$f^R\left(\sum_i r_i x_i\right) = \sum_i r_i f(x_i).$$

- For sets X and Y, we have $\sigma : (R \otimes X) \times (R \otimes Y) \to R \otimes (X \times Y)$ defined by

$$\sigma\left(\sum_i r_i x_i, \sum_j s_j y_j\right) = \sum_i \sum_j r_i s_j (x_i, y_j).$$

2.4 Adding rewards to non-deterministic systems

A common way to think of a discrete-time system is as a *decision process*. We think of the system A as an *agent* who needs to make a decision. The agent can choose an *action*, an element of In_A, and will then transition into a new state — although it may not know precisely which. We then ask the question: what is the best action for the agent to take in a given situation?

Clearly, an answer to this question will depend on what it means for one action to be better than another. The most common way to model this is by associating each action with a real number *reward*. The bigger the reward, the better the action (and negative rewards are harmful actions). If the agent is going to take a sequence of actions, we want the rewards to accumulate so that the total reward of a sequence of actions is the sum of each reward.

We can handle this accumulation of rewards, even in a deterministic system, with a commutative monad.

Definition 2.41. Let $(R, +, 0)$ be a commutative monoid (such as the real numbers). The *R-valued reward monad* or *monad of R-actions* is defined by the following data:

- To each set A, we associate the set $R \times A$ of pairs of a reward and an element of A.
- For each set A, we have $\eta_A : A \to R \times A$ given by yielding no reward: $\eta_A(a) = (0, a)$.
- For a function $f : A \to R \times B$ which yields an element of B and a reward, we give the function

$$f^R : R \times A \to R \times B$$

defined by $f^R(r, a) = (r + \pi_1 f(a), \pi_2 f(a))$. This accumulates the reward $\pi_1 f(a)$ from applying f to a onto a current reward r

- For sets A and B, we have

$$\sigma : (R \times A) \times (R \times B) \to R \times (A \times B)$$

given by $\sigma((r, a), (r', b)) = (r + r', (a, b))$. The reward for doing two actions simultaneously is the sum of their rewards.

We remark that this works not only in the category of sets, but in any cartesian category.

Exercise 2.42. Show that the monad of R-valued rewards is really a commuativative monad. That is, show that the above data satisfies all each of the laws in Definition 2.7. Do you see where the commutativity comes into the mix? ◊

We can then describe a system with reward as having an update $\text{update}_S : \text{State}_S \times \text{In}_S \to R \times \text{State}_S$ which sends the current state and action to the next state together with the reward for taking that action (in that state).

Definition 2.43. A deterministic system with R-valued rewards is an $(R \times -)$-system in the sense of Definition 2.33

We would really like to mix our rewards with non-determinism. In particular, when thinking of a system as an agent making decisions with imperfect information of its environment, we would like to use stochastic systems to model this lack of perfect information. The agent doesn't know exactly what will happen when it performs an action, but it has a good idea of what will *probably* happen.

The reward our agent gets should depend on what state the agent actually ends up in, and not just the action it takes. Therefore, we want to know the probability of transitioning to a next state *and* getting a certain reward. This has signature

$$\text{State}_S \times \text{In}_S \to \mathrm{D}(\mathbb{R} \times \text{State}_S).$$

We will show that the assignment $A \mapsto \mathrm{D}(\mathbb{R} \times A)$ forms a commutative monad. We will show that more generally, if M is *any* commutative monad and R any commutative monoid, then $M(R \times -)$ is a commutative monad again. We say that we can "put the rewards R into the monad M". We can do this explicitly using the map $\lambda : R \times MA \to M(R \times A)$ defined to be the composite

$$\lambda := R \times MA \xrightarrow{\eta^M \times \mathrm{id}} MR \times MA \xrightarrow{\sigma^M} M(R \times A)$$

Intuitively, this takes a reward $r \in R$ and a non-deterministic $a \in MA$ and gives us the non-determinstic pair (r, a).

Proposition 2.44. Let M be a commutative monad and $(R, +, 0)$ a commutative monoid. Then the assignment $A \mapsto M(R \times A)$ is a commutative monad with the following structure:

- $\eta^{M(R\times-)} : A \to M(R \times A)$ is the composite $A \xrightarrow{\eta^R} R \times A \xrightarrow{\eta^M} M(R \times A)$.
- Given $f : A \to M(R \times B)$, we define $f^{M(R\times-)}$ to be the following composite:

$$M(R \times A) \xrightarrow{M(R\times f)} M(R \times M(R \times B)) \xrightarrow{M\lambda} MM(R \times R \times B) \xrightarrow{\mu^M} M(R \times R \times B) \xrightarrow{M\mu^R} M(R \times B).$$

  Intuitively, this takes a non-deterministic pair (r, a) and, gets the non-deterministic pair $f(a) = (f_1(a), f_2(a))$, and then returns the non-deterministic pair $(r+f_1(a), f_2(a))$.
- Given sets A and B, we define $\sigma^{M(R\times-)} : M(R\times A) \to M(R\times B)$ to be the composite

$$M(R \times A) \xrightarrow{M} (R \times B) \xrightarrow{\sigma^M} M((R \times A) \times (R \times B)) \xrightarrow{M\sigma^R} M(R \times A \times B).$$

Proof. It is not obvious that this will satsify the monad laws, but it is a rather straightforward check using the laws of M and $R \times -$. We will not prove this result explicitly. However, we will give a slick proof for experts.

A monad structure on $M(R\times A)$ arising via a distributive law such as $\lambda : R\times MA \to M(R\times A)$ is equivalent to a lift of the monad M to the category of $R\times-$ algebras — that is, the category of R-actions. But $M : \mathcal{C} \to \mathcal{C}$ is a commutative monad, and so in particular it is a symmetric monoidal functor; therefore, it preserves commutative monoids and their actions. For this reason, M extends to the category of $(R \times -)$-algebras, giving us the desired monad structure on $M(R \times -)$. This is again commutative as it is the composite of monoidal functors and so also monoidal. □

Example 2.45. Let's see what this general theorem looks like in the case that $R = \mathbb{R}$ and $M = \mathrm{D}$. In this case, $\lambda : \mathbb{R} \times \mathrm{D}A \to \mathrm{D}(\mathbb{R} \times A)$ sends the pair (r, p) of a reward and a probability distribution and yields the probability distribution $\delta_r p$. Let's see how this lets us iterate the dynamics of a $\mathrm{D}(\mathbb{R} \times -)$-system S. We have $\mathsf{update}_\mathsf{S} : \mathsf{State}_\mathsf{S} \times \mathsf{In}_\mathsf{S} \to \mathrm{D}(\mathbb{R} \times \mathsf{State}_\mathsf{S})$, giving us a probabilities $\mathsf{update}_\mathsf{S}(s, i)(r, s')$ of transitioning from state s on action i into state s' and receiving reward r. To iterate this, we form the composite

$$\mathrm{D}(\mathbb{R} \times \mathsf{State}_\mathsf{S}) \times \mathsf{In}_\mathsf{S} \xrightarrow{\sigma\circ(\mathrm{id}\times\eta)} \mathrm{D}(\mathbb{R} \times \mathsf{State}_\mathsf{S} \times \mathsf{In}_\mathsf{S}) \xrightarrow{\mathsf{update}_\mathsf{S}^{\mathrm{D}(\mathbb{R}\times-)}} \mathrm{D}(\mathbb{R} \times \mathsf{State}_\mathsf{S})$$

which sends a pair (p, i) of a prior probability distribution on states and an an action to the distribution $(r, s) \mapsto \sum_{s'\in\mathsf{State}_\mathsf{S}} p(s')\mathsf{update}_\mathsf{S}(s', i)(r, s)$ which gives the probability of receiving the reward r and transitioning into the state s conditioned upon the prior p. To iterate, we can continually apply this map to many inputs; let's just do i and j.

Then we end up with the distribution

$$(r, s) \mapsto \sum_{s'' \in \mathsf{States}} \sum_{s' \in \mathsf{States}} \sum_{r''+r'=r} p(s) \cdot \mathrm{update}_{\mathsf{S}}(s'', i)(r'', s') \cdot \mathrm{update}_{\mathsf{S}}(s', j)(r', s)$$

which is the probability that we transition to s in two steps and receive a cumulative reward of r.

2.5 Changing the flavor of non-determinism: Monad maps

In the same way that 0 is a number, determinstic systems are non-deterministic systems, just with a trivial sort of non-determinism. Deterministic systems are M-systems for the *identity monad* $\mathrm{id}(X) = X$. No matter what kind of non-determinism we are considering, we can always consider a deterministic system as a non-deterministic system, because we can take the update : $\mathsf{State} \times \mathsf{In} \to \mathsf{State}$ and post compose by $\eta : \mathsf{State} \to M\mathsf{State}$. This operation of turning a deterministic system into an M-system has a few nice properties; for example, if we iterate the system and then turn it into an M-system, we get the same result as if we had iterated it as an M-system.

In general, if we have a commutative monad morphism $M \to N$, then we can turn M-systems into N-systems.

Definition 2.46. A commutative monad map $\phi : M \to N$ is a natural transformation for which the following diagrams commute:

- $$\begin{array}{ccc} A & \xrightarrow{\eta^M} & MA \\ & \searrow^{\eta^N} & \downarrow{\scriptstyle \phi} \\ & & NA \end{array} \tag{2.47}$$
- $$\begin{array}{ccc} M^2A & \xrightarrow{M\phi \fatsemi \phi} & N^2A \\ {\scriptstyle \mu^M}\downarrow & & \downarrow{\scriptstyle \mu^N} \\ MA & \xrightarrow[\phi]{} & NA \end{array} \tag{2.48}$$
- $$\begin{array}{ccc} MA \times MB & \xrightarrow{\phi \times \phi} & NA \times NB \\ {\scriptstyle \sigma^M}\downarrow & & \downarrow{\scriptstyle \sigma^N} \\ M(A \times B) & \xrightarrow[\phi]{} & N(A \times B) \end{array} \tag{2.49}$$

Proposition 2.50. There is a unique commutative monad map $\mathrm{id} \to M$, and it is given by η^M.

Proof. Let ϕ be such a map. Then condition Eq. (2.47) says precisely that $\phi = \eta^M$. So it just remains to check that η is a commutative monad map. Now, Eq. (2.47) commutes

trivially, and Eq. (2.48) is in this case one of the diagrams defining M from Eq. (2.8). Finally, Eq. (2.49) is in this case Eq. (2.12). □

We can then turn any deterministic system S into an M-system by defining its new update to be $\eta^M \circ \text{update}_S$. For possibilistic systems, this says that only the state that S actually transitions into is possible. For stochastic systems, this says that the probability that the system transitions into the state it actually transitions into is 1.

Intuitively, stochastic non-determinism is a refinement of possibilistic non-determinism: it not only tells us what is possible, but how likely it is. We can package this intuition into a commutative monad morphism $\phi : \mathrm{D} \to \mathrm{P}$.

Proposition 2.51. There is a commutative monad morphism $\phi : \mathrm{D} \to \mathrm{P}$ given by sending a probability distribution to the set of elements with non-zero probability:

$$\phi(p) = \{a \in A \mid p(a) \neq 0\}.$$

Proof. We check that this satisfies the laws.

- (Eq. (2.47)) The only element which δ_a assigns a non-zero probability is a.
- (Eq. (2.48)) Given a formal convex combination $\sum_i \lambda_i p_i$ of probability distributions $p_i \in \mathrm{D}A$, we see that

  $$\phi\mu^{\mathrm{D}}\left(\sum_i \lambda_i p_i\right) = \{a \in A \mid \sum_i \lambda_i p_i(a) \neq 0\},$$

  while

  $$\mathrm{D}\phi\left(\sum_i \lambda_i p_i\right) = \sum_i \lambda_i \{a \in A \mid p_i(a) \neq 0\}$$

  and so taking ϕ of that yields

  $$\{\{a \in A \mid p_i(a) \neq 0\} \mid \lambda_i \neq 0\}$$

  so, finally

  $$\mu^{\mathrm{P}}\left(\phi\mathrm{D}\phi\left(\sum_i \lambda_i p_i\right)\right) = \bigcup_{\lambda_i \neq 0} \{a \in A \mid p_i(a) \neq 0\}.$$

  Both paths around the square are equal since all of the λ_i and $p_i(a)$ are positive.
- (Eq. (2.49)) Let p be a probability distribution on A and q a probability distribution on B. Then

  $$\phi(\sigma(p, q)) = \{(a, b) \mid p(a)q(b) \neq 0\}$$

  while

  $$\sigma(\phi(p), \phi(q)) = \{(a, b) \mid p(a) \neq 0\, q(b) \neq 0\}.$$

  These are equal since $p(a)q(b) \neq 0$ if and only if both $p(a)$ and $q(b)$ are not 0.

□

This lets us turn a stochastic system into a possibilistic system, saying that a transition is possible if it has non-zero probability.

Exercise 2.52. Show that $\mathrm{D}\eta^{\mathbb{R}} : \mathrm{D}A \to \mathrm{D}(\mathbb{R} \times A)$ is a commutative monad morphism. That is, show that the following diagrams commute:

1.
$$\begin{array}{ccc} A & \xrightarrow{\eta^{\mathrm{D}}} & \mathrm{D}A \\ & {\scriptstyle \eta^{\mathrm{D}(\mathbb{R}\times -)}} \searrow & \downarrow {\scriptstyle \mathrm{D}(\eta^{\mathbb{R}})} \\ & & \mathrm{D}(\mathbb{R} \times A) \end{array} \tag{2.53}$$

2.
$$\begin{array}{ccc} \mathrm{D}^2 A & \xrightarrow{\mathrm{D}\mathrm{D}\eta^{\mathbb{R}} \fatsemi \mathrm{D}\eta^{\mathbb{R}}} & \mathrm{D}(\mathbb{R} \times \mathrm{D}(\mathbb{R} \times A)) \\ {\scriptstyle \mu^{\mathrm{D}}}\downarrow & & \downarrow {\scriptstyle \mu^{\mathrm{D}(\mathbb{R}\times -)}} \\ \mathrm{D}A & \xrightarrow[\mathrm{D}\eta^{\mathbb{R}}]{} & \mathrm{D}(\mathbb{R} \times A) \end{array} \tag{2.54}$$

3.
$$\begin{array}{ccc} \mathrm{D}A \times \mathrm{D}B & \xrightarrow{\mathrm{D}\eta^{\mathbb{R}} \times \mathrm{D}\eta^{\mathbb{R}}} & \mathrm{D}(\mathbb{R} \times A) \times \mathrm{D}(\mathbb{R} \times B) \\ {\scriptstyle \sigma^{\mathrm{D}}}\downarrow & & \downarrow {\scriptstyle \sigma^{\mathrm{D}(\mathbb{R}\times -)}} \\ \mathrm{D}(A \times B) & \xrightarrow[\mathrm{D}\eta^{\mathbb{R}}]{} & \mathrm{D}(\mathbb{R} \times A \times B) \end{array} \tag{2.55}$$

This shows that we can always consider a stochastic system as a stochastic system with rewards by assigning every transition the reward 0. ◊

The reason we need all the laws for the monad morphism and not just an arbitrary family of maps $\phi : MA \to NA$ is that with these laws, we get functors $\mathbf{Kl}(M) \to \mathbf{Kl}(N)$ which tell us that iterating and then changing our non-determinism is the same as changing our non-determinism and then iterating. We begin with a useful lemma.

Lemma 2.56. In the definition of a commutative monad map $\phi : M \to N$, the commutativity of diagram Eq. (2.48) can be replaced by the commutativity of the following diagram for any $f : A \to MB$:

$$\begin{array}{ccc} MA & \xrightarrow{\phi} & NA \\ {\scriptstyle f^M}\downarrow & & \downarrow {\scriptstyle (f \fatsemi \phi)^N} \\ MB & \xrightarrow[\phi]{} & NB \end{array} \tag{2.57}$$

That is,

$$f^M \fatsemi \phi = \phi \fatsemi (f \fatsemi \phi)^N.$$

Proof. Before we begin, we note that, by the naturality of ϕ, $M\phi \fatsemi \phi = \phi \fatsemi N\phi$:

$$\begin{array}{ccc} M^2A & \xrightarrow{\phi_{MA}} & NMA \\ {\scriptstyle M\phi_A}\downarrow & & \downarrow{\scriptstyle N\phi_A} \\ MNA & \xrightarrow[\phi_{NA}]{} & N^2A \end{array}$$

That is, we can take the top of Eq. (2.48) to be $\phi \fatsemi N\phi$ rather than $M\phi \fatsemi \phi$.

We recall that $f^M = Mf \fatsemi \mu^M$, and similarly $(f \fatsemi \phi)^N = N(f \fatsemi \phi) \fatsemi \mu^N$. So we may rewrite Eq. (2.57) as the solid outer diagram in

$$\begin{array}{ccc} MA & \xrightarrow{\phi} & NA \\ {\scriptstyle Mf}\downarrow & & \downarrow{\scriptstyle Nf\fatsemi N\phi)} \\ M^2B & \overset{\phi\fatsemi N\phi}{\dashrightarrow} & N^2B \\ {\scriptstyle \mu^M}\downarrow & & \downarrow{\scriptstyle \mu^N} \\ MB & \xrightarrow[\phi]{} & NB \end{array} \tag{2.58}$$

Now we are ready to prove our lemma. We note that the top square in this diagram always commutes by the naturality of ϕ. Eq. (2.48) is the lower square in this diagram; so, if it commutes, then the outer square (which is Eq. (2.57)) commutes. On the other hand, if Eq. (2.57) commmutes for all $f : A \to MB$, we may take $f = \mathrm{id} : MA \to MA$ to find that the outer square of Eq. (2.58) becomes just Eq. (2.48). □

Proposition 2.59. Let $\phi : M \to N$ be a commutative monad morhpisms. Then there is a strict symmetric monoidal functor

$$\phi_* : \mathbf{Kl}(M) \to \mathbf{Kl}(N)$$

acting as the identity on objects and sending the Kleisli map $f : A \to MB$ to the composite

$$\phi_* f \coloneqq A \xrightarrow{f} MB \xrightarrow{\phi} NB.$$

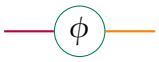
Proof. We will check that this is a functor; that it is strictly symmetric monoidal follows from this and from the fact that it acts as the identity on objects. The identity $\eta^M : A \to MA$ in $\mathbf{Kl}(M)$ gets sent to $\phi_*\eta^M = \eta^M \fatsemi \phi$. This equals $\eta^N : A \to NA$ by Eq. (2.47).

Given $f : A \to MB$ and $g : B \to MC$, their composite is $f \fatsemi g^M : A \to MC$, so that

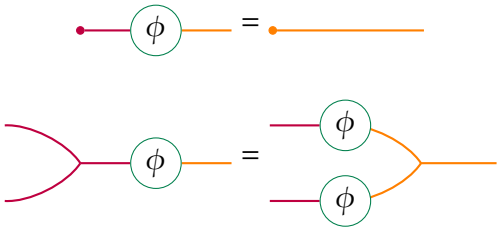
$$\begin{aligned} \phi_*(f \fatsemi g^M) &\coloneqq f \fatsemi g^M \fatsemi \phi \\ &= f \fatsemi \phi \fatsemi (g \fatsemi \phi)^N && \text{by Lemma 2.56} \\ &= (\phi_* f)(\phi_* g)^N. \end{aligned}$$

□

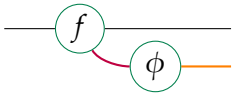
We can also check that ϕ_* is a functor using our string diagram notation for monads. In that notation, $\phi : M \to N$ is written as



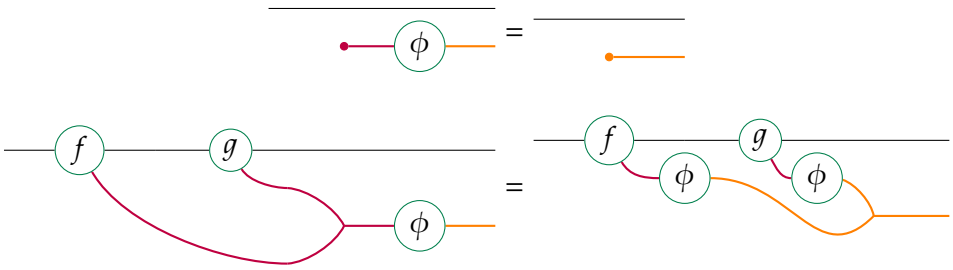
and would satisfy the laws:



(As before, these diagrams are not really equipped to describe the commutativity of monads, and so we are only using the laws concerning the unit and multiplication.) The action of ϕ_* on a Kleisli map $f : X \to MY$ is then written as



We can check that ϕ_* is functorial quickly and diagrammatically:

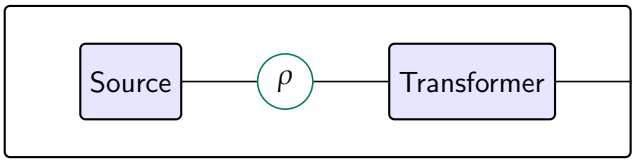


2.6 Wiring together non-deterministic systems: the generalized lens construction

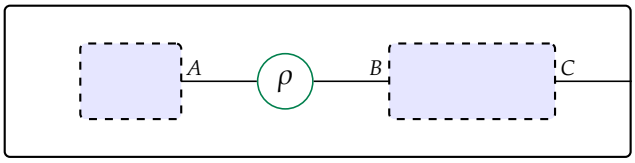
Consider a stochastic source process

We can imagine, as Claude Shannon did, that this source is an interlocutor communicating over a wire. Suppose we have another interlocutor who will read the signal generated by our source and generate their own signal in repsonse:

Having these two models, we can form a new stochastic source by considering them together:



We imagine that the Transformer listens to the signal generated by the Source, but with noise ρ on the wire. This wiring diagram



can be described as a *monadic lens* $\binom{f^\sharp}{f} : \binom{B}{A\times C} \leftrightarrows \binom{1}{C}$:

- $f : A \times C \to C$ is the projection π_2.
- $f^\sharp : A \times C \times 1 \to DB$ is $\rho \circ \pi_1$ where $\rho : A \to DB$ is the stochastic function describing the noise on the wire.

This new notion of monadic lens which lets us wire together non-deterministic systems will be the focus of this section.

In Section 1.3, we saw how to wire together systems deterministically and with functions from an algebraic theory on the wire. This worked because wiring diagrams could be interpreted as lenses, and deterministic and differential systems were also lenses; then we could just compose them.

But non-deterministic systems are not lenses in a cartesian category; they have that monad sitting over the states in the codomain of update:

$$\mathsf{update}_\mathsf{S} : \mathsf{State}_\mathsf{S} \times \mathsf{In}_\mathsf{S} \to M\mathsf{State}_\mathsf{S}.$$

It may appear that we could consider this as a map in the Kleisli category, and just take lenses in the Kleisli category. But in the Kleisli category, the operation $\times$ is rarely a cartesian product, and we can only describe lenses in cartesian categories. The reason we can only describe lenses in cartesian categories is because in the formula for the passback of a composite of lenses, we use a variable twice; that is, we use the diagonal map $\Delta : A^+ \to A^+ \times A^+$, a feature of cartesian categories.

We will need a new perspective on lenses and lens composition which suggests how to change the passback of the lenses. It is worth noting that we only need to duplicate in the passforward direction; we should be free to change the passback direction.

In this section, we will give a new perspective on the category of lenses using the *Grothendieck construction*. This perspective constructs the category of lenses out of an *indexed category* $\mathbf{Ctx}_\_ : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ of objects of the cartesian category $\mathcal{C}$ *in context*. This construction works for *any* indxed category $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$, which lets us define a notion of $\mathcal{A}$-lenses out of any indexed category. By choosing an appropriate indexed

category, we will arrive at the notion of M-lenses for a commutative monad M; this will give us the wiring diagram calculus for non-deterministic systems that we wanted.

First, we introduce the abstract categorical notions of *indexed category* and the *Grothendieck construction*.

2.6.1 Indexed categories and the Grothendieck construction

An indexed category $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ is a family of categories $\mathcal{A}(C)$ that varies functorially with an object $C \in \mathcal{C}$ of the *base category* $\mathcal{C}$ (see **??** for the full definition). We will intepret the base category $\mathcal{C}$ as the category of passforward maps, and the categories $\mathcal{A}(C^+)$ as the categories of passback maps that take C^+ as an extra argument.

Definition 2.60. A *strict indexed category* $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ is a contravariant functor[a]. We call the category $\mathcal{C}$ the *base* of the indexed category $\mathcal{A}$. Explicitly, an indexed category $\mathcal{A}$ has:

- A base category $\mathcal{C}$.
- For every object $C \in \mathcal{C}$ of the base, a category $\mathcal{A}(C)$.
- For every map $f : C \to C'$ in the base, a *pullback* functor $f^* : \mathcal{A}(C') \to \mathcal{A}(C)$, which we think of as "reindexing" the objects of $\mathcal{A}(C')$ so that they live over $\mathcal{A}(C)$.
- Reindexing is functorial: $(f \fatsemi g)^* = g^* \fatsemi f^*$ and $\mathrm{id}^* = \mathrm{id}$.

[a] A pseudo-functor is like a functor, but it only satisfies the functoriality conditions up to isomorphism, and these isomorphisms must satisfy some laws that make them "cohere". The indexed categories we will see in Chapters 1 to 3 will all be actual functors, however.

Remark 2.61. We have given the definition of a *strict* indexed category. A general indexed category is a *pseudo*-functor $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$, which is like a functor but functoriality only holds up to coherent isomorphism. As in the case of monoidal categories, the coherences in the isomorphisms are often just bookkeeping trivialities.

However, the theory of strict indexed categories is noticeably easier, and most of our examples will be strict. Since we will mostly be using strict indexed categories, we will often refer to them simply as "indexed categories".

Indexed categories are quite common throughout mathematics. We will construct a particular example for our own purposes in Section 3.4, and more throughout the book.

Example 2.62. Recall that a *dependent set* is a function $X : A \to \mathbf{Set}$ from a set into the category of sets. We have an indexed category of dependent sets

$$\mathbf{Set}^{(-)} : \mathbf{Set}^{\mathrm{op}} \to \mathbf{Cat}$$

which is defined as follows:

- To each set A, we assign the category $\mathbf{Set}^A$ of sets indexed by A. The objects of

$\mathbf{Set}^A$ are the sets $X : A \to \mathbf{Set}$ indexed by A, and a map $f : X \to Y$ is a family of maps $f_a : X_a \to Y_a$ indexed by the elements $a \in A$. Composition is given componentwise: $(g \circ f)_a = g_a \circ f_a$.

- To every function $f : A' \to A$, we get a reindexing functor

$$f^* : \mathbf{Set}^A \to \mathbf{Set}^{A'}$$

Given by precomposition: $X \mapsto X \circ f$. The indexed set $X \circ f : A' \to \mathbf{Set}$ is the set $X_{f(a')}$ on the index $a' \in A'$. The families of functions get reindexed the same way.

- Since our reindexing is just given by precomposition, it is clearly functorial.

We will return to this example in much greater detail in **??**.

If we have an family of sets $A : I \to \mathbf{Set}$ indexed by a set I, we can form the disjoint union $\sum_{i \in I} A_i$, together with the projection $\pi : \sum_{i \in I} A_i \to I$ sending each $a \in A_i$ to i. The Grothendieck construction is a generalization of this construction to indexed categories. Namely, we will take an indexed category $\mathcal{A} : \mathcal{C} \to \mathbf{Cat}$ and form a new category

$$\int^{C:\mathcal{C}} \mathcal{A}(C)$$

which we think of as a "union" off all the categories $\mathcal{A}(C)$. But this "union" will not be disjoint since there will be morphisms from objects in $\mathcal{A}(C)$ to objects in $\mathcal{A}(C')$. This is why we use the integral notation; we want to suggest that the Grothendieck construction is a sort of sum.[2]

Definition 2.63. Let $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ be an indexed category. The *Grothendieck construction* of $\mathcal{A}$

$$\int^{C:\mathcal{C}} \mathcal{A}(C)$$

is the category with:

- Objects pairs $\begin{pmatrix} A \\ C \end{pmatrix}$ of objects $C \in \mathcal{C}$ and $A \in \mathcal{A}(C)$. We say that A "sits over" C.
- Maps $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} A \\ C \end{pmatrix} \rightrightarrows \begin{pmatrix} A' \\ C' \end{pmatrix}$ pairs of $f : C \to C'$ in $\mathcal{C}$ and $f_\flat : A \to f^*A'$ in $\mathcal{A}(C)$.
- Given $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} A \\ C \end{pmatrix} \rightrightarrows \begin{pmatrix} A' \\ C' \end{pmatrix}$ and $\begin{pmatrix} g_\flat \\ g \end{pmatrix} : \begin{pmatrix} A' \\ C' \end{pmatrix} \rightrightarrows \begin{pmatrix} A'' \\ C'' \end{pmatrix}$, their composite is given by

$$\begin{pmatrix} g_\flat \\ g \end{pmatrix} \circ \begin{pmatrix} f_\flat \\ f \end{pmatrix} := \begin{pmatrix} f^* g_\flat \circ f_\flat \\ g \circ f \end{pmatrix}$$

[2] The Grothendieck construction is an example of a *lax colimit* in 2-category theory, another sense in which it is a 'sort of sum'.

Written with the signatures, this looks like

$$\begin{pmatrix} A \xrightarrow{f_\flat} f^*A' \xrightarrow{f^*g_\flat} f^*g^*A'' = (g \circ f)^*A'' \\ C \xrightarrow{f} C' \xrightarrow{g} C'' \end{pmatrix}$$

- The identity is given by $\begin{pmatrix} \mathrm{id}_A \\ \mathrm{id}_C \end{pmatrix} : \begin{pmatrix} A \\ C \end{pmatrix} \rightrightarrows \begin{pmatrix} A \\ C \end{pmatrix}$

Exercise 2.64. Check that Definition 3.22 does indeed make $\int^{C:\mathcal{C}} \mathcal{A}(C)$ into a category. That is, check that composition as defined above is associative and unital. ◊

The confluence of notation with that of charts is no accident. Indeed, we will see in the next section that the category of charts can be described as the Grothendieck construction of a certain indexed category.

2.6.2 Maps with context and lenses

In this section, we'll see the category $\mathbf{Lens}_\mathcal{C}$ of lenses in a cartesian category $\mathcal{C}$ can be described using the Grothendieck construction. To do this, we need some other categories named after their maps (rather than their objects): *category of maps with context* C for some a given $C \in \mathcal{C}$.

Definition 2.65. Let $\mathcal{C}$ be a cartesian category and let $C \in \mathcal{C}$. The *category* $\mathbf{Ctx}_C$ *of maps with context* C is the category defined by:

- Objects are the objects of $\mathcal{C}$.
- Maps $f : X \rightsquigarrow Y$ are maps $f : C \times X \to Y$.
- The composite $g \circ f$ of $f : X \rightsquigarrow Y$ and $g : Y \rightsquigarrow Z$ is the map

$$(c, x) \mapsto g(c, f(c, x)) : C \times X \to Z.$$

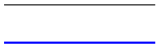
  Diagrammatically, this is the composite:

$$C \times X \xrightarrow{\Delta_C \times X} C \times C \times X \xrightarrow{C \times f} C \times Y \xrightarrow{g} Z.$$

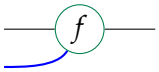
- The identity $\mathrm{id} : X \rightsquigarrow X$ is the second projection $\pi_2 : C \times X \to X$.

We can prove that $\mathbf{Ctx}_C$ is a category using a similar string diagrams to those we used in Section 2.3. We have a functor $X \mapsto C \times X : \mathcal{C} \to \mathcal{C}$ which we can draw as a blue string:

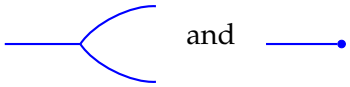
If we represent $X \in \mathcal{C}$ by the string  then we represent $C \times X$ as



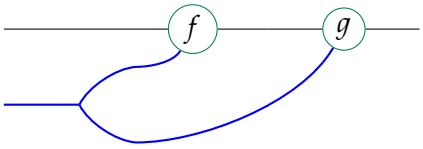
We can therefore represent a morphism $f : C \times X \to Y$ in the context of C as a bead like this:



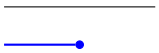
To compose maps in context, we need the diagonal map $\Delta_C \times X : C \times X \to C \times C \times X$ and the second projection $\pi_2 : C \times X \to X$. Since these maps are natural in X, we can draw them as



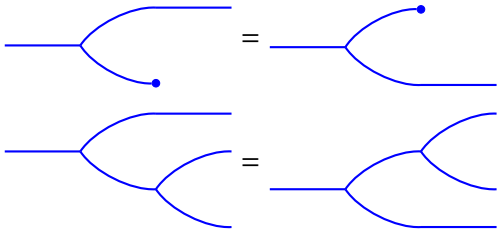
Then the composition in $\mathbf{Ctx}_C$ of maps in context $f : C \times X \to Y$ and $g : C \times Y \to Z$ is drawn as:



and the second projection $\pi_2 : C \times X \to X$ is drawn



This is exactly dual to the story about Kleisli composition we saw in Section 2.3! To show that $\mathbf{Ctx}_C$ is a category, we need to note that the following equations hold:



These say that

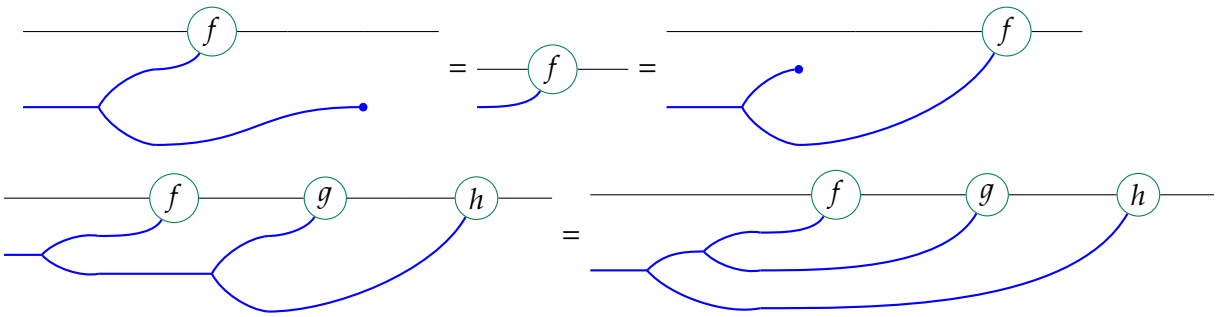
$$(\Delta_C \times X) \fatsemi (\pi_2 \times X) = \mathrm{id}_{C \times X} = (\Delta_C \times X) \fatsemi (C \times \pi_2)$$

and

$$(\Delta_C \times X) \fatsemi (C \times \Delta_C \times X) = (\Delta_C \times X) \fatsemi (\Delta_C \times X \times C).$$

These hold by some simple work in the cartesian category $\mathcal{C}$ (see Exercise 2.66).

With these laws in hand, we can prove associativity and identity of composition in $\mathbf{Ctx}_C$ by appealing to the following diagrams:



Exercise 2.66. Show that the following composites are equal in any cartesian category:

1.
$$(\Delta_C \times X) \fatsemi (\pi_2 \times X) = (\Delta_C \times X) \fatsemi (C \times \pi_2)$$
These are both maps $C \times X \to C \times C \times C \times X$.

2.
$$(\Delta_C \times X) \fatsemi (C \times \Delta_C \times X) = \mathrm{id}_{C \times X} = (\Delta_C \times X) \fatsemi (\Delta_C \times X \times C).$$
These are all maps $C \times X \to C \times X$.

◊

Exercise 2.67. Show that $\mathbf{Ctx}_1$ is equivalent to the underlying cartesian category $\mathcal{C}$. In other words, maps in the context 1 have "no context". ◊

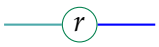
Together, we can arrange the categories of maps with context into an indexed category.

Definition 2.68. The *indexed category of maps with context*
$$\mathbf{Ctx}_\_ : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$$
is defined by:

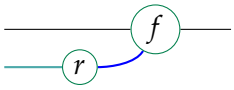
- For $C \in \mathcal{C}$, we have the category $\mathbf{Ctx}_C$ of maps with context C.
- For a map $r : C' \to C$, we get a reindexing functor
$$r^* : \mathbf{Ctx}_C \to \mathbf{Ctx}_{C'}$$
given by sending each object to itself, but each morphism $f : C \times X \to Y$ in $\mathbf{Ctx}_C$ to the map $r^* f := f \circ (r \times X)$:
$$C' \times X \xrightarrow{r \times X} C \times X \xrightarrow{f} Y.$$
On elements,
$$r^* f(c', x) := f(r(c'), x).$$

We note that this is evidently functorial.

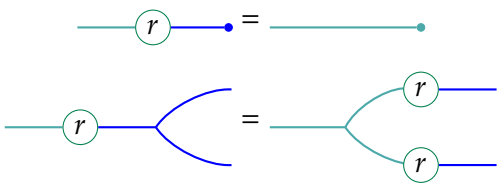
To see that to every $r : C' \to C$ we get a functor $r^* : \mathbf{Ctx}_C \to \mathbf{Ctx}_{C'}$, we can use string diagrams. We can draw r as



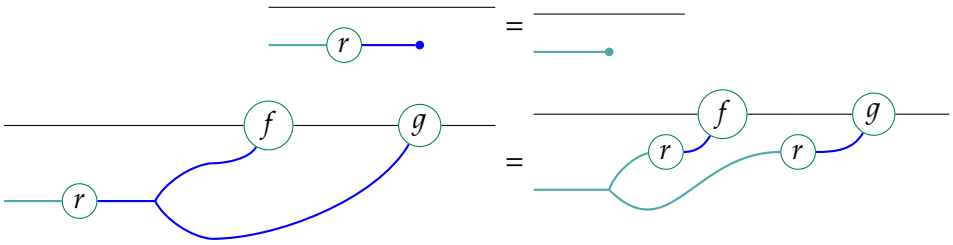
so that the action of r^* is given by



If we note that r satisfies the following laws:



we can then prove that r^* is a functor graphically:



Proposition 2.69. The category $\mathbf{Lens}_{\mathcal{C}}$ of lenses in $\mathcal{C}$ is the Grothendieck construction of the indexed category of *opposites* of the categories of maps with context:

$$\mathbf{Lens}_{\mathcal{C}} = \int^{C \in \mathcal{C}} \mathbf{Ctx}_C^{\mathrm{op}}.$$

Proof. We will expand the definition of the right hand side and see that it is precisely the category of lenses.

The objects of $\int^{C \in \mathbf{Set}} \mathbf{Ctx}_C^{\mathrm{op}}$ are pairs $\begin{pmatrix} A^- \\ A^+ \end{pmatrix}$ of objects of $\mathcal{C}$. All good so far.

A map in $\int^{C \in \mathcal{C}} \mathbf{Ctx}_C^{\mathrm{op}}$ is a pair $\begin{pmatrix} f^\sharp \\ f \end{pmatrix}$ with $f : A^+ \to B^+$ and $f^\sharp : A^- \rightsquigarrow f^* B^-$ in $\mathbf{Ctx}_{A^+}^{\mathrm{op}}$. Now, $f^* B^- = B^-$ so $f^\sharp$ has signature $A^- \rightsquigarrow B^-$ in $\mathbf{Ctx}_{A^+}^{\mathrm{op}}$, which means $f^\sharp$ has signature $B^- \rightsquigarrow A^-$ in $\mathbf{Ctx}_{A^+}$, which means that $f^\sharp$ is a really a function $A^+ \times B^- \to A^-$. In other words, a map in $\int^{C \in \mathbf{Set}} \mathbf{Ctx}_C^{\mathrm{op}}$ is precisely a lens. We note that the identity map is the identity lens.

Finally, we need to check that composition in $\int^{C \in \mathcal{C}} \mathbf{Ctx}_C^{\mathrm{op}}$ is lens composition. Suppose that $\begin{pmatrix} f^\sharp \\ f \end{pmatrix} : \begin{pmatrix} A^- \\ A^+ \end{pmatrix} \leftrightarrows \begin{pmatrix} B^- \\ B^+ \end{pmatrix}$ and $\begin{pmatrix} g^\sharp \\ g \end{pmatrix} : \begin{pmatrix} B^- \\ B^+ \end{pmatrix} \leftrightarrows \begin{pmatrix} C^- \\ C^+ \end{pmatrix}$ are lenses. In $\int^{C \in \mathbf{Set}} \mathbf{Ctx}_C^{\mathrm{op}}$, their composite is

$$\begin{pmatrix} f^* g^\sharp \circ f^\sharp \\ g \circ f \end{pmatrix}.$$

The bottom is all good, we just need to check that the top — which, remember, lives in $\mathbf{Ctx}_{A^+}^{\mathrm{op}}$ — is correct. Since the composite up top is in the opposite, we are really calculating $f^\sharp \circ f^* g^\sharp$ in $\mathbf{Ctx}_{A^+}$. By definition, this is

$$(a^+, c^-) \mapsto f^\sharp(a^+, g^\sharp(f(a^+), c^-))$$

which is precisely their composite as lenses! □

Exercise 2.70. Make sure you *really* understand Proposition 3.28. ◊

We take Proposition 3.28 as paradigmatic of the notion of lens, and use this idea to define lenses from any indexed category.

Definition 2.71. Let $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ be an indexed category. The category of $\mathcal{A}$-lenses is the Grothendieck construction of $\mathcal{A}^{\mathrm{op}}$:

$$\mathbf{Lens}_{\mathcal{A}} = \int^{C \in \mathcal{C}} \mathcal{A}(C)^{\mathrm{op}}.$$

Example 2.72. Recall the indexed category $\mathbf{Set}^{(-)} : \mathbf{Set}^{\mathrm{op}} \to \mathbf{Cat}$ of dependent sets from Example 3.21. A $\mathbf{Set}^{(-)}$-lens $\begin{pmatrix} f^\sharp \\ f \end{pmatrix} : \begin{pmatrix} A^-_a \\ a \in A^+ \end{pmatrix} \leftrightarrows \begin{pmatrix} B^-_b \\ b \in B^+ \end{pmatrix}$ consists of

- A passforward function $f : A^+ \to B^+$, and
- A family of passback functions $f^\sharp_a : B^- \to A^-$ for every $a \in A^+$.

We call these *dependent lenses*, and they will be a major character in the later chapters of this book.

2.6.3 Monoidal indexed categories and the product of lenses

To describe wiring diagrams, it is not enough just to have the category of lenses; we also need the monoidal product

$$\begin{pmatrix} A^- \\ A^+ \end{pmatrix} \otimes \begin{pmatrix} B^- \\ B^+ \end{pmatrix} \coloneqq \begin{pmatrix} A^- \times B^- \\ A^+ \times B^+ \end{pmatrix}.$$

We need this product to put systems together before wiring them. In order to wire together non-deterministic systems, we will need to generalize this product of lenses to generalized lenses. For this, we will need the notion of an *monoidal indexed category* and the associated *monoidal Grothendieck construction* as defined in [**MoellerVasilakopolou**].

Definition 2.73. A *monoidal strict indexed category* $(\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}, \otimes, 1, \boxtimes, \maltese)$ consists of:

- A strict indexed category $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$,
- A monoidal structure $(\otimes, 1)$ on $\mathcal{C}$,
- A natural family of functors $\boxtimes : \mathcal{A}(C) \times \mathcal{A}(C') \to \mathcal{A}(C \otimes C')$ and $\maltese \in \mathcal{A}(1)$ with natural isomorphisms

$$A_1 \boxtimes (A_2 \boxtimes A_3) \cong (A_1 \boxtimes A_2) \boxtimes A_3,$$

$$\maltese \boxtimes A \cong A \cong A \boxtimes \maltese.$$

These natural isomorphisms are required to satisfy coherences reminiscent of

those of a monoidal category.

Theorem 2.74 ([**MoellerVasilakopolou**])**.** Let $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ be a monoidal indexed category. Then the Grothendieck construction $\int^{C:\mathcal{C}} \mathcal{A}(C)$ may be equipped with a monoidal structure

$$\begin{pmatrix} A^- \\ A^+ \end{pmatrix} \otimes \begin{pmatrix} B^- \\ B^+ \end{pmatrix} \coloneqq \begin{pmatrix} A^- \boxtimes B^- \\ A^+ \otimes B^+ \end{pmatrix}.$$

If the base of indexing $\mathcal{C}$ is cartesian, then there is a simpler way to describe a monoidal structure on an indexed category $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$.

Theorem 2.75 ([**ShulmanMonoidal**])**.** Let $\mathcal{C}$ be a cartesian category. Then a monoidal structures on a strict indexed category $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ whose underlying monoidal structure on $\mathcal{C}$ is given by the cartesian product may be equivalently given by the data:

- A monoidal structure $\otimes : \mathcal{A}(C) \times \mathcal{A}(C) \to \mathcal{A}(C)$ and $1 \in \mathcal{A}(C)$ for each $C \in \mathcal{C}$,
- A lax structure on each reindexing $r^* : \mathcal{A}(C) \to \mathcal{A}(C')$ for each $r : C' \to C$, so that the lax structure on $(r_2 \circ r_1)^*$ is the composite of the lax structures on r_2 and r_1.

Proof Sketch. We define the product $\otimes : \mathcal{A}(C) \times \mathcal{A}(C) \to \mathcal{A}(C)$ as $\boxtimes \circ \Delta^*$ where $\Delta : C \to C \times C$ is the diagonal. We similarly define $1 \in \mathcal{A}(C)$ as $!^*(\not\vdash)$. □

We use Theorem 2.75 and Theorem 2.74 to recover the product of lenses.

Lemma 2.76. Let $\mathcal{C}$ be a cartesian category and let $C \in \mathcal{C}$. The category $\mathbf{Ctx}_C$ has a monoidal structure given by $X \otimes Y \coloneqq X \times Y$, $1 \coloneqq 1$, and

$$f \otimes g \coloneqq C \times X \times Y \xrightarrow{\Delta} C \times C \times X \times Y \xrightarrow{\sim} C \times X \times C \times Y \xrightarrow{f \times g} X' \times Y'.$$

In terms of elements,

$$(f \otimes g)(c, x, y) \coloneqq (f(c, x), g(c, y)).$$

Proof. We begin by showing that $\otimes$ is functorial:

$$\begin{aligned} ((f' \circ f) \otimes (g' \circ g))(c, x, y) &= ((f' \circ f)(c, x), (g' \circ g)(c, y)) \\ &= (f'(c, f(c, x)), g'(c, g'(c, y))) \\ &= (f' \otimes g')(f(c, x), g(c, y)) \\ &= (f' \otimes g') \circ (f \otimes g)(x, y). \end{aligned}$$

Next, we need associators $X \otimes (Y \otimes Z) \cong (X \otimes Y) \otimes Z$ and unitors $1 \otimes X \cong X \cong X \otimes 1$. We may get these by applying $!^* : \mathcal{C} \to \mathbf{Ctx}_C$ (which sends $f : X \to Y$ to $f \circ \pi_1 : C \times X \to Y$) to the associators and unitors of $\mathcal{C}$. It is straightforward to see that these are natural with respect to maps in $\mathbf{Ctx}_C$. □

Proposition 2.77. Let $\mathcal{C}$ be a cartesian category. Then $\mathbf{Ctx}_- : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ may be endowed with a monoidal structure so that the induced monoidal structure on the Grothendieck construction is the product of lenses

$$\begin{pmatrix} A^- \\ A^+ \end{pmatrix} \otimes \begin{pmatrix} B^- \\ B^+ \end{pmatrix} := \begin{pmatrix} A^- \times B^- \\ A^+ \times B^+ \end{pmatrix}.$$

Proof. By Lemma 2.76, there is monoidal structure on each $\mathbf{Ctx}_C$. We note that by definition, each reindexing $r^* : \mathbf{Ctx}_C \to \mathbf{Ctx}_{C'}$ along $r : C' \to C$ preserves this monoidal structure strictly.

$$\begin{aligned} r^*(f \otimes g)(c', (x, y)) &= (f \otimes g)(r(c'), (x, y)) \\ &= (f(r(c'), x), g(r(c'), y)) \\ &= (r^* f \otimes r^* g)(c, (x, y)). \end{aligned}$$

The rest then follows by Theorem 2.75 and Theorem 2.74. □

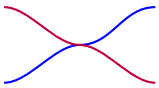
2.6.4 Monadic lenses as generalized lenses

Now we are ready to define monadic lenses. We have a formula: we just need to find the right indexed category. We will do this by modifying the definition of $\mathbf{Ctx}_C$ so that a map is of the form $C \times X \to MY$. If the resulting categories $\mathbf{Ctx}_C^M$ remain indexed over $\mathcal{C}$, we have a ready made notion of monadic lens and monadic lens composition given by the Grothendieck construction!

We will be able to define composition in the categories $\mathbf{Ctx}_C^M$ by making use of the natural map

$$\lambda : C \times MX \xrightarrow{\eta \times MX} MC \times MX \xrightarrow{\sigma} M(C \times X)$$

Using string diagrams, we may draw this map as



Definition 2.78. Let $\mathcal{C}$ be a cartesian category and $M : \mathcal{C} \to \mathcal{C}$ a commutative monad. For an object $C \in \mathcal{C}$, there is a category $\mathbf{Ctx}_C^M$ with:

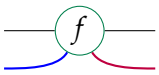
- Objects the objects of $\mathcal{C}$.
- Map $f : X \rightsquigarrow Y$ are maps $f : C \times X \to MY$ in $\mathcal{C}$.
- The identity $X \rightsquigarrow X$ is $\pi_2 \fatsemi \eta$.
- The composite $f \fatsemi g$ of $f : X \rightsquigarrow Y$ and $g : Y \rightsquigarrow Z$ is given by

$$f \fatsemi g := (\Delta_C \times X) \fatsemi (C \times f) \fatsemi \lambda \fatsemi Mg \fatsemi \mu.$$

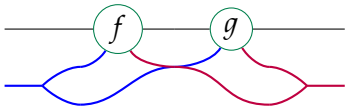
$$C \times X \to C \times C \times X \to C \times MY \to M(C \times Y) \to M^2 Z \to MZ$$

Here, $\lambda := (\eta \times MX) \fatsemi \sigma$.

We can show that $\mathbf{Ctx}_C^M$ is indeed a category using string diagrams. In string diagrams, a map $f : C \times X \to MY$ in $\mathbf{Ctx}_C^M$ is drawn

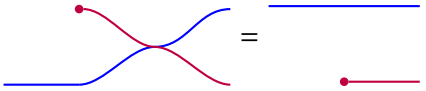


and composition is drawn

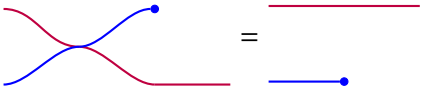


The identity is drawn

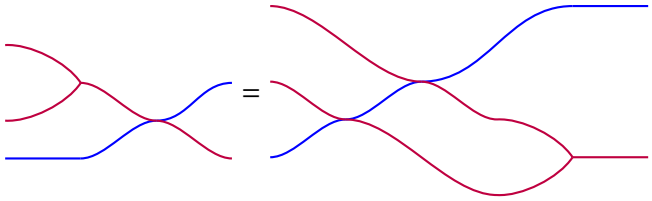
In order to show that this composition is unital and associative, we will need to show that the following four laws hold relating λ to the structure of M and of $C \times (-)$:



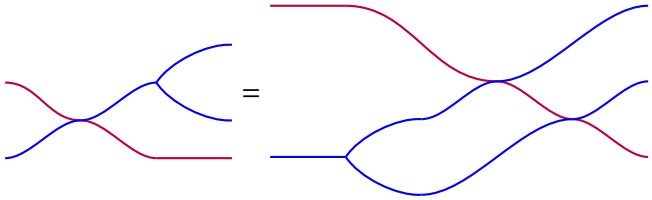
(2.79)



(2.80)

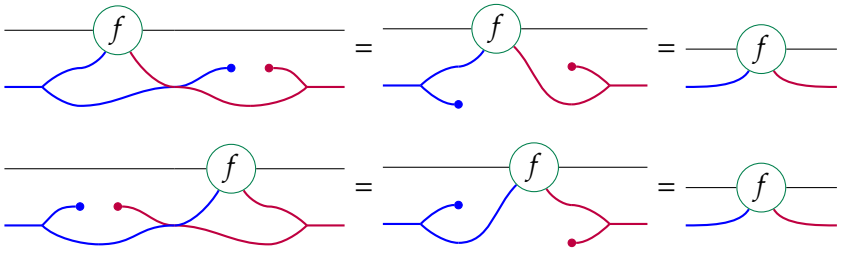


(2.81)

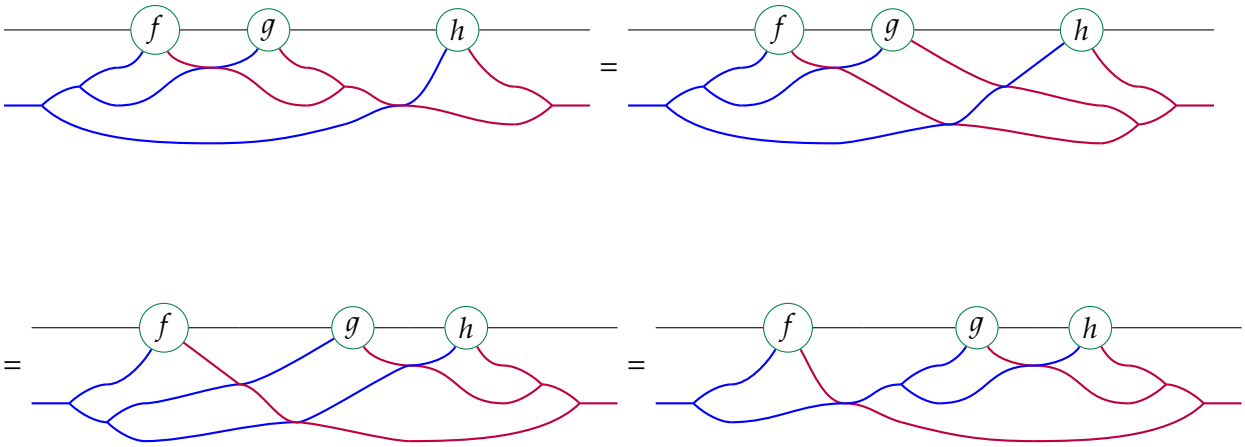


(2.82)

We will prove these laws in the upcoming Lemma 2.84.[3] Using them, we can see that composition in $\mathbf{Ctx}_C^M$ is unital and associative.



[3] And we will re-express them as commutative diagrams there.



This shows that $\mathbf{Ctx}_C^M$ is a category. We furthermore want an indexed category $\mathbf{Ctx}_-^M : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$, so that we can apply the generalized lens construction to it and get a category of M-lenses.

Theorem 2.83. Let $M : \mathcal{C} \to \mathcal{C}$ be a commutative monad on a cartesian category. Then there is an indexed category

$$\mathbf{Ctx}_-^M : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$$

which sends an object $C \in \mathcal{C}$ to the category $\mathbf{Ctx}_C^M$ and which sends a map $r : C' \to C$ to the functor

$$r^* : \mathbf{Ctx}_C^M \to \mathbf{Ctx}_{C'}^M$$

which acts as the identity on objects and which sends a morphism $f : C \times X \to MY$ to the composite $C' \times X \xrightarrow{r \times X} C \times X \xrightarrow{f} MY$.

We can show that the reindexing r^* given by $r : C' \to C$ as defined above is indeed a functor by appealing to the naturality of λ in C. This indexed category is in fact a strict functor (as opposed to a pseudo-functor), as can be seen right away from the definition of the reindexing functors.

We now return to prove the crucial laws which undergird the above graphical arguments.

Lemma 2.84. Let $M : \mathcal{C} \to \mathcal{C}$ be a commutative monad on a cartesian category $\mathcal{C}$. Then the map $\lambda : C \times MX \to M(C \times X)$ defined by

$$\lambda := (\eta \times MX) \fatsemi \sigma$$

is natural in both X and C. Furthermore, the following four diagrams commute:

$$\begin{array}{ccc} C \times X & \xrightarrow{C \times \eta} & C \times MX \\ & \searrow^{\eta} & \downarrow{\scriptstyle \lambda} \\ & & M(C \times X) \end{array} \tag{2.85}$$

$$\begin{array}{ccc} C \times MX & \xrightarrow{\lambda} & M(C \times X) \\ & \searrow^{\pi_2} & \downarrow{\scriptstyle M\pi_2} \\ & & MX \end{array} \tag{2.86}$$

$$\begin{array}{ccccc} C \times M^2X & & \xrightarrow{\quad C \times \mu \quad} & & C \times MX \\ \downarrow{\scriptstyle \lambda} & & & & \downarrow{\scriptstyle \lambda} \\ M(C \times MX) & \xrightarrow[M\lambda]{} & M^2(C \times X) & \xrightarrow[\mu]{} & M(C \times X) \end{array} \tag{2.87}$$

$$\begin{array}{ccccc} C \times MX & & \xrightarrow{\quad \lambda \quad} & & M(C \times X) \\ \downarrow{\scriptstyle \Delta_C \times MX} & & & & \downarrow{\scriptstyle M(\Delta_C \times X)} \\ C \times C \times MX & \xrightarrow[C \times \lambda]{} & C \times M(C \times X) & \xrightarrow[\lambda]{} & M(C \times C \times X) \end{array} \tag{2.88}$$

Exercise 2.89. Prove Lemma 2.84 by showing that the diagrams commute. This uses the properties of the commutativity σ and naturality. ◊

In order to wire diagrams together, we also need the monoidal product on lenses.

Lemma 2.90. Let $\mathcal{C}$ be a cartesian category and let $M : \mathcal{C} \to \mathcal{C}$ be a commutative monad. Then for any $C \in \mathcal{C}$, there is a symmetric monoidal structure on $\mathbf{Ctx}_C^M$ given by $X \otimes Y := X \times Y$ and

$$f \otimes g := C \times X \times Y \xrightarrow{\Delta} C \times C \times X \times Y \xrightarrow{\sim} C \times X \times C \times Y \xrightarrow{f \times g} MX \times MY \xrightarrow{\sigma} M(X \times Y).$$

Proof. □

2.7 Changing the Flavor of Non-determinism

In Section 2.5, we saw how commutative monad maps $\phi : M \to N$ let us change the flavor of non-determinism. In particular, since the unit $\eta : \mathrm{id} \to M$ is always a commutative monad map, we can always interpret a deterministic system as a non-deterministic system.

In this section, we'll show that any commutative monad morphism $\phi : M \to N$ induces a

2.8 Monadic Lenses

Chapter 2

How systems behave: Trajectories, Steady States, and Periodic Orbits

2.1 Introduction

So far, we have seen how to wire up dynamical systems. But we haven't seen our dynamical systems actually *do anything*. In this section, we will begin to study the behavior of our dynamical systems. We will see particular kinds of behaviors our systems can have: trajectories, steady states, and periodic orbits.

Informal Definition 2.1. A *behavior* of a dynamical system is a particular way its states can change according to its dynamics.

There are different *kinds of behavior* corresponding to the different sorts ways that the states of a system could evolve. Perhaps they eventually repeat, or they stay the same despite changing conditions.

In Section 2.5, we will give a formal definition of behavior of dynamical system. We will see that the different kinds of behaviors — trajectories, steady states, periodic orbits, etc. — can each be packaged up into a single system that *represents* that kind of behavior. This system will behave in exactly that kind of way, and do nothing else. Maps from it to a system of interest will exhibit that sort of behavior in the system of interest.

We will end the section by seeing how the steady states of a complex system formed by wiring together some component systems can be calculated from the steady states of the components. The calculation will turn out to be just a bit of matrix arithmetic!

This result will be a preview to our more general results in **??** which concern arbitrary behaviors of dynamical systems. But that's getting ahead of ourselves.

2.2 Trajectories

In the introduction, we saw that the Clock system Eq. (1.7) has behaves in this way if it starts at 3 o'clock:

$$3 \overset{\text{tick}}{\longmapsto} 4 \overset{\text{tick}}{\longmapsto} 5 \overset{\text{tick}}{\longmapsto} 6 \overset{\text{tick}}{\longmapsto} \cdots$$

This sequence of states of the clock system, each following from the last by the dynamics of the system, is called a *trajectory*. When our systems have input parameters, we will need to choose a sequence of input parameters to feed the system in order for the states to change.

Definition 2.2. Let

$$\mathsf{S} = \begin{pmatrix} \mathrm{update}_{\mathsf{S}} \\ \mathrm{expose}_{\mathsf{S}} \end{pmatrix} : \begin{pmatrix} \mathrm{State}_{\mathsf{S}} \\ \mathrm{State}_{\mathsf{S}} \end{pmatrix} \leftrightarrows \begin{pmatrix} \mathrm{In}_{\mathsf{S}} \\ \mathrm{Out}_{\mathsf{S}} \end{pmatrix}$$

be a deterministic system. Suppose that $p : \mathbb{N} \to \mathrm{In}_{\mathsf{S}}$ is a sequence of parameters for S. Then a *p-trajectory* of S is a sequence $s : \mathbb{N} \to \mathrm{State}_{\mathsf{S}}$ of states so that

$$\mathrm{update}_{\mathsf{S}}(s_i, p_i) = s_{i+1}$$

for all $i \in \mathbb{N}$.

If additionally $v : \mathbb{N} \to \mathrm{Out}_{\mathsf{S}}$ is a sequence of output values for S, then a $\binom{p}{v}$*-trajectory* is a sequence of states $s : \mathbb{N} \to \mathrm{State}_{\mathsf{S}}$ so that

$$\mathrm{update}_{\mathsf{S}}(s_i, p_i) = s_{i+1}$$
$$\mathrm{expose}_{\mathsf{S}}(s_i) = v_i$$

for all $i \in \mathbb{N}$. We call the pair $\binom{p}{v}$ the *chart* of the trajectory s.

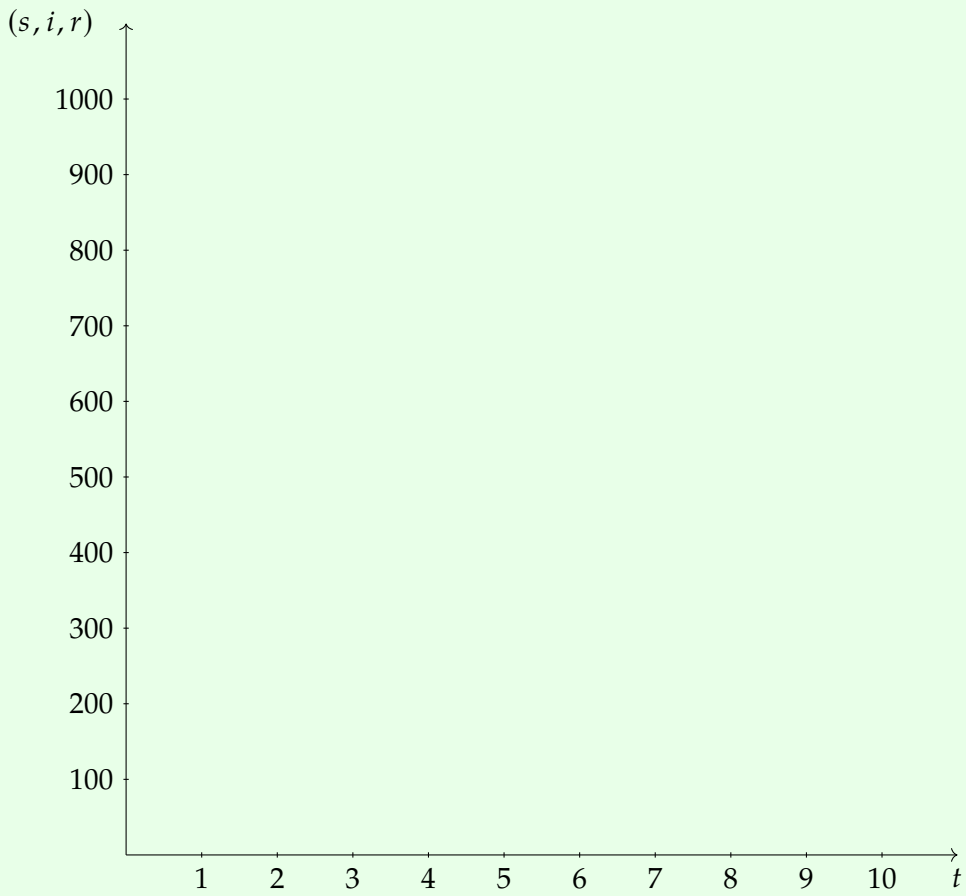
Its worth noting that a trajectory $s : \mathbb{N} \to \mathrm{State}_{\mathsf{S}}$ in a deterministic system is determined entirely by its start state s_0. This is what makes deterministic systems deterministic; if you know the dynamics and you know what state the system is in, you know how it will continue to behave. We'll relax this condition later in **??**.

Example 2.3. Consider the SIR model of Example 1.25. Suppose that we let our parameters $(a, b) : \mathbb{N} \to \mathrm{In}_{\mathsf{SIR}}$ be constant at .2 and .3 respectively: that is, $a_t = .2$ and $b_t = .3$ for all t. Then a trajectory for SIR with parameters (a, b) is a sequence of populations $(s, i, r) : \mathbb{N} \to \mathrm{State}_{\mathsf{SIR}}$ such that

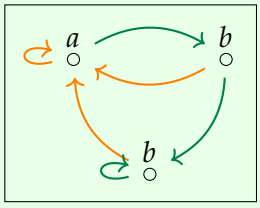
$$\begin{bmatrix} s_{t+1} \\ i_{t+1} \\ r_{t+1} \end{bmatrix} = \begin{bmatrix} s_t - .2 s_t i_t \\ i_t + .2 s_t i_t - .3 i_t \\ r_t + .3 i_t \end{bmatrix}$$

Here is an example of such a trajectory with a 1000 total people and one infected

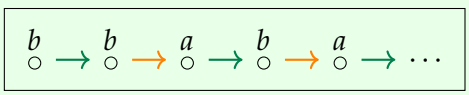
person to start, that is $(s_0, i_0, r_0) = (999, 1, 0)$.



Example 2.4. If a deterministic system is written as a transition diagram, then the trajectories in the system are paths through the diagram. Recall from Example 1.28 this system:



Suppose that $p : \mathbb{N} \to \{\texttt{green}, \texttt{orange}\}$ alternates between `green` and `orange`. Then starting at the top right state, a trajectory quickly settles into alternating between the top two states:



Knowing about trajectories can show us another important role that deterministic systems play: they are *stream transformers*. From a stream $p : \mathbb{N} \to \mathsf{In}_S$ of inputs and a start state $s_0 \in \mathsf{State}_S$, we get a trajectory $s : \mathbb{N} \to \mathsf{State}_S$ given recursively by

$$s_{t+1} := \mathrm{update}_S(s_t, p_t).$$

We then get a stream $v : \mathbb{N} \to \mathsf{Out}_S$ of output values by defining

$$v_t := \mathrm{expose}_S(s_t).$$

The system S is a way of transforming streams of input parameters into streams of output values.

Proposition 2.5 (Deterministic systems as stream transformers)**.** Let

$$S = \begin{pmatrix} \mathrm{update}_S \\ \mathrm{expose}_S \end{pmatrix} : \begin{pmatrix} \mathsf{State}_S \\ \mathsf{State}_S \end{pmatrix} \leftrightarrows \begin{pmatrix} \mathsf{In}_S \\ \mathsf{Out}_S \end{pmatrix}$$

be a deterministic system. Then for every $s_0 \in \mathsf{State}_S$, we get a stream transformation function

$$\mathrm{transforms}_S : \mathsf{In}_S^{\mathbb{N}} \to \mathsf{Out}_S^{\mathbb{N}}$$

Given by

$$\mathrm{transforms}_S(p)_0 = \mathrm{expose}_S(s_0)$$
$$\mathrm{transforms}_S(p)_{t+1} = \mathrm{expose}_S(\mathrm{update}_S(s_t, p_t))$$

where $s_{t+1} = \mathrm{update}_S(s_t, p_t)$ is the trajectory given by s_0.

Exercise 2.6. Say how the system of Example 2.4 acts as a stream transformer on the following streams:

1. $p_{2t} =$ `green` and $p_{2t+1} =$ `orange`.
2. $p_t =$ `green`.
3. $p_0 =$ `green` and $p_t =$ `orange` for all $t > 0$. ◊

Later, in **??**, we will see that given trajectories of component systems, we get a trajectory of a whole wired system. Even better, every trajectory of the whole wired system can be calculated this way.

2.3 Steady states

A steady state of a system is a state which does not change. Steady states are important because they are guarantees of stability: a vase in a steady state is doing great, a heart in a steady state is in need of attention.

Definition 2.7. Let

$$\mathsf{S} = \begin{pmatrix} \mathrm{update}_{\mathsf{S}} \\ \mathrm{expose}_{\mathsf{S}} \end{pmatrix} : \begin{pmatrix} \mathrm{State}_{\mathsf{S}} \\ \mathrm{State}_{\mathsf{S}} \end{pmatrix} \leftrightarrows \begin{pmatrix} \mathrm{In}_{\mathsf{S}} \\ \mathrm{Out}_{\mathsf{S}} \end{pmatrix}$$

be a deterministic system. For input parameter $i \in \mathrm{In}_{\mathsf{S}}$ and output value $o \in \mathrm{Out}_{\mathsf{S}}$, an $\binom{i}{o}$-*steady state* is a state $s \in \mathrm{State}_{\mathsf{S}}$ such that

$$\mathrm{update}_{\mathsf{S}}(s, i) = s,$$
$$\mathrm{expose}_{\mathsf{S}}(s) = o.$$

We call the pair $\binom{i}{o}$ the *chart* of the steady state.

Remark 2.8. Its important to note that a steady state is relative to the input parameter chosen. For example, in Example 1.28, the top left state is steady for the input parameter `orange` but not for the input parameter `green`.

Unlike with trajectories, a system need not necessarily have *any* steady states. For example, the Clock has no steady states; it always keeps ticking to the next hour.

In the transition diagram of a finite deterministic system, steady states will be loops that begin and end at the same node. Since the system is finite, we can arrange the steady states by their chart into a $\mathrm{In}_{\mathsf{S}} \times \mathrm{Out}_{\mathsf{S}}$ matrix. For example, in Example 1.28, we get the following $\{\texttt{green}, \texttt{orange}\} \times \{a, b\}$ matrix:

$$\begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{cc} \texttt{green} & \texttt{orange} \end{array} \\ \begin{bmatrix} \emptyset & \{\circlearrowright\ a\} \\ \{\circlearrowright\ b\} & \emptyset \end{bmatrix} \end{array} \tag{2.9}$$

This is a "matrix of sets", in that the entries are the actual sets of steady states. If we just counted how many steady states there were for each input-output pair, we would get this matrix:

$$\begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{cc} \texttt{green} & \texttt{orange} \end{array} \\ \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{array} \tag{2.10}$$

In Section 3.2, we'll see that each wiring diagram gives a formula for calculating the matrix of steady states of the composite system from the matrices of steady states of the inner systems.

Exercise 2.11. What are the steady state matrices of systems S_1 and S_2 from Exercise 1.67? What about the combined system S? ◊

Steady-looking trajectories. The reason we are interested in steady states is that they are highly predictable; if we know we are in a steady state, then we know we are always going to get the same results. But it is possible for us to always get the same outputs for the same input even though the internal state keeps changing. These are special trajectories, and we call them *steady-looking* trajectories.

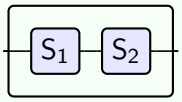
Definition 2.12. For $i \in \mathsf{In}_\mathsf{S}$ and $o \in \mathsf{Out}_\mathsf{S}$ of a system S, a $\binom{i}{o}$-*steady looking trajectory* is a sequence of states $s : \mathbb{N} \to \mathsf{State}_\mathsf{S}$ such that

$$\mathrm{update}_\mathsf{S}(s_t, i) = s_{t+1}$$
$$\mathrm{expose}_\mathsf{S}(s_t) = o$$

for all $t \in \mathbb{N}$. We call the pair $\binom{i}{o}$ the *chart* of the steady-looking trajectory s.

While the steady states of a wired together system can be calculated from those of its components, this is not true for steady-looking trajectories. Intuitively, this is because the internal systems can be exposing changing outputs between eachother even while the eventual external output remains unchanged.

Exercise 2.13. Consider the wiring diagram:

$\mathsf{S} :=$ 

Find systems S_1 and S_2 and a steady-looking trajectory of the wired system S which is not steady-looking on the component systems. ◊

2.4 Periodic orbits

Even if the behavior of a system isn't perfectly steady, it may continually repeat. To a reasonable approximation, the position of the earth around the sun follows a cycle that repeats every year. Using this as a paradigmatic example, we call these behaviors that repeat *periodic orbits*.

Definition 2.14 (Periodic orbit)**.** A $\binom{p}{v}$-trajectory $s : \mathbb{N} \to \mathsf{State}_\mathsf{S}$ is *periodic* if there exists a time $t_0 \in \mathbb{N}_{\geq 1}$, called the *period*, such that $s_{t_0} = s_0$. If the sequence of parameters $p : \mathbb{N} \to \mathsf{In}_\mathsf{S}$ is also periodic with the same period (in that $p_{t_0} = p_0$ as well), then we say that s has *periodic parameters*.

Remark 2.15. Note that when we say that a periodic orbit has periodic parameters, we assume that they are periodic with the same period. This has important but subtle

consequences for our theorems concerning the composition of behaviors in **??**. We explain the difference between a periodic orbit and a periodic orbit with periodic parameters in a more precise manner in Remark 2.32.

Remark 2.16. Note that a steady state is a periodic orbit (with periodic parameters) that has a period of 1.

Exercise 2.17. Describe a periodic orbit with period 1 that does not have periodic parameters; how are they different from steady states? Are there any of these in systems S_1 and S_2 of Exercise 1.67? ◊

Example 2.18. The Clock system is an exemplary periodic system with a period of 12. The ClockWithDisplay of Eq. (1.11) has period 24.

Exercise 2.19. What are the periodic orbits in the systems S_1 and S_2 of Exercise 1.67 with periodic parameters, and what are their periods? What about the combined system S? ◊

Exercise 2.20. Can you think of any periodic orbits in S_1 and S_2 of Exercise 1.67 which don't have periodic parameters? ◊

A trajectory might not get back to where it started, but may still end up being periodic. We call these trajectories *eventually* periodic orbits, since they eventually end up in a repeating cycle of states.

Definition 2.21 (Eventually periodic orbit)**.** A $\binom{p}{v}$-trajectory $s : \mathbb{N} \to \mathsf{State}_\mathsf{S}$ is *eventually periodic* if there are times $t_0 < t_1 \in \mathbb{N}$ such that $s_{t_0+t} = s_{t_1+t}$ for all $t \in \mathbb{N}$. If the sequence of parameters $p : \mathbb{N} \to \mathsf{In}_\mathsf{S}$ is also eventually periodic with the same period (in that $p_{t_0+t} = p_{t_1+t}$ for all t), then we say that s has *eventually periodic parameters*.

The *period* of an eventually periodic trajectory is the smallest difference $t_1 - t_0$ between times such that $s_{t_0} = s_{t_1}$.

2.5 Behaviors of deterministic systems

In the previous Sections 2.2 and 2.4 and **??**, we saw a number of different kinds of behaviors of dynamical systems. Not only were there a lot of definitions in those sections, each of those definitions had slight variants (like periodic orbits versus periodic orbits with periodic parameters, or steady states versus steady-looking trajectories). In this section, we'll define a general notion of behavior and see that we can package each of

the above sorts of behavior into a single system in its own right, one that *represents* that sort of behavior. The representative system of a certain kind of behavior behaves in exactly that way, and does nothing else.

To get started, we will give a formal definition of behavior of deterministic system. Before we do this, we need to define another sort of map that can go between the interfaces of systems.

Definition 2.22 (Category of charts)**.** A *chart* $\binom{f_\flat}{f} : \binom{A^-}{A^+} \rightrightarrows \binom{B^-}{B^+}$ in a cartesian category $\mathcal{C}$ is a pair of maps $f : A^+ \to B^+$ and $f_\flat : A^+ \times A^- \to B^-$. Note that *this is not a lens*.

Exercise 2.23.

1. How many lenses are there $\binom{f^\sharp}{f} : \binom{3}{2} \leftrightarrows \binom{4}{3}$?
2. How many charts are there $\binom{f^\flat}{f} : \binom{3}{2} \rightrightarrows \binom{4}{3}$? ◊

Definition 2.24 (Behavior of deterministic systems)**.** Let T and S be deterministic systems. Given a chart of interfaces $\binom{f_\flat}{f} : \binom{\mathsf{In}_\mathsf{T}}{\mathsf{Out}_\mathsf{T}} \rightrightarrows \binom{\mathsf{In}_\mathsf{S}}{\mathsf{Out}_\mathsf{S}}$, a $\binom{f_\flat}{f}$*-behavior of shape* T *in* S, written $\phi : \mathsf{T} \to \mathsf{S}$, is a function $\phi : \mathsf{State}_\mathsf{T} \to \mathsf{State}_\mathsf{S}$ sending states of T to states of S which preserves the dynamics and exposed variables by satisfying the following equations:

$$\begin{aligned} \mathrm{expose}_\mathsf{S}(\phi(t)) &= f(\mathrm{expose}_\mathsf{T}(t)), \\ \mathrm{update}_\mathsf{S}(\phi(t), f_\flat(\mathrm{expose}_\mathsf{T}(t), i)) &= \phi(\mathrm{update}_\mathsf{T}(t, i)) \end{aligned} \tag{2.25}$$

for all $t \in \mathsf{State}_\mathsf{T}$ and $i \in \mathsf{In}_\mathsf{T}$. We say that $\binom{f_\flat}{f}$ is the chart of the behavior ϕ.

Remark 2.26. If you prefer commutative diagrams to systems of equations, don't fret. We'll reinterpret Eq. (2.25) in terms of commutative diagrams in **??**

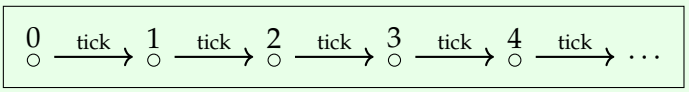
Remark 2.27. Suppose that we have transition diagrams for systems T and S. Then a behavior of shape T in S will correspond to part of the transition diagram of S which is shaped like the transition diagram of T. See the upcoming examples to see how this looks in practice.

Let's make this definition feel real with a few examples.

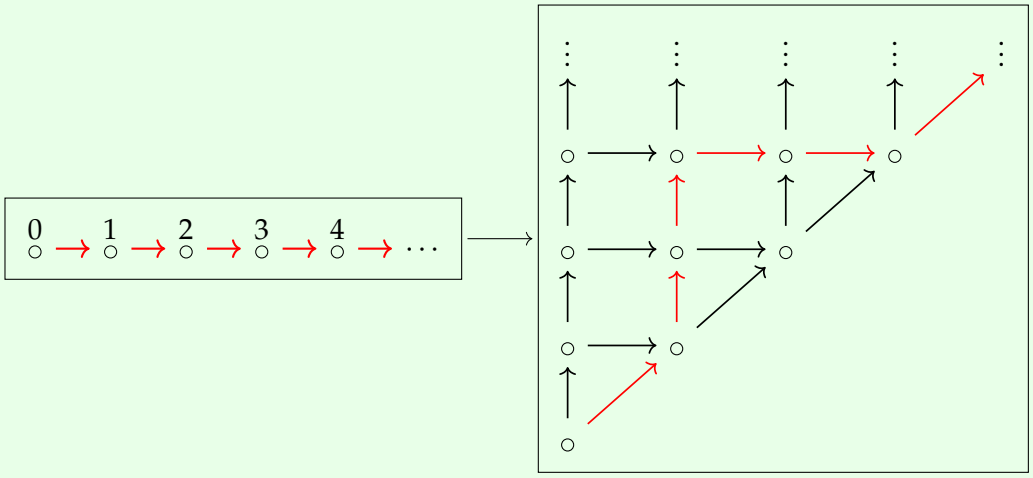
Example 2.28. Let Time be the system $\binom{t \mapsto t+1}{\mathrm{id}} : \binom{\mathbb{N}}{\mathbb{N}} \leftrightarrows \binom{\{\mathrm{tick}\}}{\mathbb{N}}$, i.e. with

- $\mathsf{State}_\mathsf{Time} := \mathbb{N}$,
- $\mathsf{Out}_\mathsf{Time} := \mathbb{N}$,
- $\mathsf{In}_\mathsf{Time} := \{\mathrm{tick}\}$,
- $\mathrm{expose}_\mathsf{Time} = \mathrm{id}$,
- $\mathrm{update}_\mathsf{Time}(t, *) = t + 1$.

As a transition diagram, Time looks like this:



Let's see what a behavior of shape Time in an arbitrary system S will be. We will expect the shape of Time to appear in the transition diagram of S, like this:



First, we need to know what a chart $\binom{f_\flat}{f} : \binom{\mathsf{In}_{\mathsf{Time}}}{\mathsf{Out}_{\mathsf{Time}}} \rightrightarrows \binom{\mathsf{In}_{\mathsf{S}}}{\mathsf{Out}_{\mathsf{S}}}$ is like. Since $\mathsf{Out}_{\mathsf{Time}} = \mathbb{N}$ and $\mathsf{In}_{\mathsf{Time}} \cong 1$, this means $f : \mathbb{N} \to \mathsf{Out}_{\mathsf{S}}$ is a sequence of outputs, and $f_\flat : \mathbb{N} \times 1 \to \mathsf{In}_{\mathsf{S}}$ is a sequence of input parameters. We might as well instead call f our sequence of exposed values v, and $f_\flat$ our sequence of input parameters p, so that we have a chart $\binom{p}{v} : \binom{1}{\mathbb{N}} \rightrightarrows \binom{\mathsf{In}_{\mathsf{S}}}{\mathsf{Out}_{\mathsf{S}}}$.

Now, let's see what a $\binom{p}{v}$-behavior $\gamma : \mathsf{Time} \to \mathsf{S}$ is. It is a function $\gamma : \mathsf{State}_{\mathsf{Time}} \to \mathsf{State}_{\mathsf{S}}$ satsifying some properties. But $\mathsf{State}_{\mathsf{Time}} = \mathbb{N}$, so $\gamma : \mathbb{N} \to \mathsf{State}_{\mathsf{S}}$ is a sequence of states in S. Now, Eq. (2.25) becomes the equations:

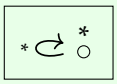
$$\begin{aligned} \mathrm{expose}_{\mathsf{S}}(\gamma(t)) &= v(t) \\ \mathrm{update}_{\mathsf{S}}(\gamma(t), p(t)) &= \gamma(t+1). \end{aligned}$$

which are exactly the equations defining a $\binom{p}{v}$-trajectory from Definition 2.2!

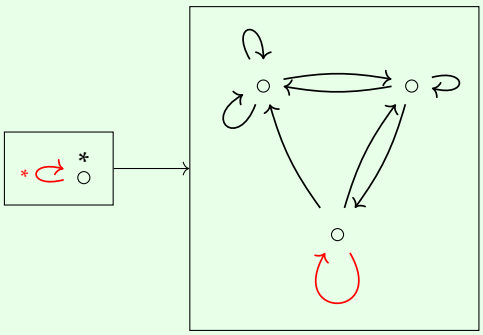
Example 2.29. Consider the simple system Fix with:

- $\mathsf{State}_{\mathsf{Fix}} = \{*\}$.
- $\mathsf{Out}_{\mathsf{Fix}} = \{*\}$.
- $\mathsf{In}_{\mathsf{Fix}} = \{*\}$.
- $\mathrm{expose}_{\mathsf{Fix}} = \mathrm{id}$.
- $\mathrm{update}_{\mathsf{Fix}}(*, *) = *$.

As a transition diagram, this looks like:



A behavior $s : \mathsf{Fix} \to \mathsf{S}$ in an arbitrary system S should be a loop of this shape within the transition diagram of S: a steady state.



Let's check that this works. First, we need to know what a chart $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} \mathsf{In}_{\mathsf{Fix}} \\ \mathsf{Out}_{\mathsf{Fix}} \end{pmatrix} \rightrightarrows \begin{pmatrix} \mathsf{In}_{\mathsf{S}} \\ \mathsf{Out}_{\mathsf{S}} \end{pmatrix}$ is. Since $\mathsf{Out}_{\mathsf{Fix}} = \mathsf{In}_{\mathsf{Fix}} = \{*\}$, we have that $f : \{*\} \to \mathsf{Out}_{\mathsf{S}}$ is simply an output value of S and $f_\flat : \{*\} \times \{*\} \to \mathsf{In}_{\mathsf{S}}$ is simply an input parameter. Therefore, we might as well write o for f and i for $f_\flat$, to see that a chart $\begin{pmatrix} i \\ o \end{pmatrix} : \begin{pmatrix} \{*\} \\ \{*\} \end{pmatrix} \rightrightarrows \begin{pmatrix} \mathsf{In}_{\mathsf{S}} \\ \mathsf{Out}_{\mathsf{S}} \end{pmatrix}$ is a pair of elements $i \in \mathsf{In}_{\mathsf{S}}$ and $o \in \mathsf{Out}_{\mathsf{S}}$.

Now, let's see what a $\begin{pmatrix} i \\ o \end{pmatrix}$-behavior $s : \mathsf{Fix} \to \mathsf{S}$ is. It is a function $s : \mathsf{State}_{\mathsf{Fix}} \to \mathsf{State}_{\mathsf{S}}$ satisfying a few properties. But $\mathsf{State}_{\mathsf{S}} = \{*\}$ so $s : \{*\} \to \mathsf{State}_{\mathsf{S}}$ is a single state of S. Then, Eq. (2.25) becomes the equations

$$\begin{aligned} \mathrm{expose}_{\mathsf{S}}(s) &= o \\ \mathrm{update}_{\mathsf{S}}(s, i) &= s \end{aligned}$$

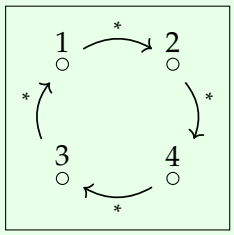
which are precisely the equations defining a $\begin{pmatrix} i \\ o \end{pmatrix}$-steady state from Definition 2.7.

Example 2.30. Let $0 < n \in \mathbb{N}$ be a positive natural number, and consider the system Clock_n having:

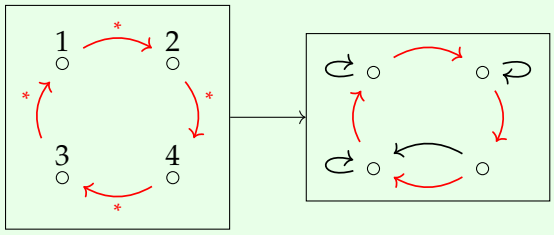
- $\mathsf{State}_{\mathsf{Clock}_n} = \mathsf{n} = \{1, \ldots, n\}$.
- $\mathsf{Out}_{\mathsf{Clock}_n} = \mathsf{n}$.
- $\mathsf{In}_{\mathsf{Clock}_n} = \{*\}$.
- $\mathrm{expose}_{\mathsf{Clock}_n} = \mathrm{id}$.
- $\mathrm{update}_{\mathsf{Clock}_n}(t, *) = \begin{cases} t+1 & \text{if } t < n \\ 1 & \text{if } t = n \end{cases}$.

This is the clock with n hours. Our example system Clock from Example 1.22 is

Clock_{12}, a clock with 12 hours. Here's what Clock_4 looks like as a transition diagram:



A behavior $\gamma : \mathsf{Clock}_\mathsf{n} \to \mathsf{S}$ should be a cycle like this in the transition diagram of S: a periodic orbit. We can see the Clock_4-behavior inside the system shown right:



Let's check that this works. First, we need to know what a chart $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} \mathsf{In}_{\mathsf{Clock}_\mathsf{n}} \\ \mathsf{Out}_{\mathsf{Clock}_\mathsf{n}} \end{pmatrix} \rightrightarrows \begin{pmatrix} \mathsf{In}_\mathsf{S} \\ \mathsf{Out}_\mathsf{S} \end{pmatrix}$ is. Since $\mathsf{Out}_{\mathsf{Clock}_\mathsf{n}} = \mathsf{n}$ and $\mathsf{In}_{\mathsf{Clock}_\mathsf{n}} = \{*\}$, $f : \mathsf{n} \to \mathsf{Out}_\mathsf{S}$ is a sequence of n exposed values of S while $f_\flat : \mathsf{n} \times \{*\} \to \mathsf{In}_\mathsf{S}$ is a sequence of n parameters. Therefore, we might as well write v for f and p for $f_\flat$ to find that a chart $\begin{pmatrix} p \\ v \end{pmatrix} : \begin{pmatrix} \{*\} \\ \mathsf{n} \end{pmatrix} \rightrightarrows \begin{pmatrix} \mathsf{In}_\mathsf{S} \\ \mathsf{Out}_\mathsf{S} \end{pmatrix}$ consists of an n-length sequence of parameters and an n-length sequence of exposed values.

A $\begin{pmatrix} p \\ v \end{pmatrix}$-behavior $\gamma : \mathsf{Clock}_\mathsf{n} \to \mathsf{S}$, then, is a function $\gamma : \mathsf{State}_{\mathsf{Clock}_\mathsf{n}} \to \mathsf{State}_\mathsf{S}$ satisfying a few properties. Since $\mathsf{State}_{\mathsf{Clock}_\mathsf{n}} = \mathsf{n}$, $\gamma : \mathsf{n} \to \mathsf{State}_\mathsf{S}$ is a n-length sequence of states of S, and Eq. (2.25) become the equations

$$\mathrm{expose}_\mathsf{S}(\gamma(t)) = v(t)$$
$$\mathrm{update}_\mathsf{S}(\gamma(t), p(t)) = \begin{cases} \gamma(t+1) & \text{if } t < n \\ \gamma(1) & \text{if } t = n \end{cases}.$$

As we can see, this determines a sequence of length n of states of S which repeats when it gets to the end. In other words, this is a periodic orbit with periodic parameters as in Definition 2.14!

If we have a certain kind of behavior in mind, and we find a system T so that behaviors of shape T are precisely this kind of behavior, then we say that T *represents* that behavior. For example, we have just seen that:

- The system $\mathsf{Time} = \begin{pmatrix} {}_{-}{}_{+1} \\ \mathrm{id} \end{pmatrix} : \begin{pmatrix} \mathbb{N} \\ \mathbb{N} \end{pmatrix} \leftrightarrows \begin{pmatrix} \{*\} \\ \mathbb{N} \end{pmatrix}$ represents trajectories.
- The system $\mathsf{Fix} = \begin{pmatrix} \pi_2 \\ \mathrm{id} \end{pmatrix} : \begin{pmatrix} \{*\} \\ \{*\} \end{pmatrix} \leftrightarrows \begin{pmatrix} \{*\} \\ \{*\} \end{pmatrix}$ represents steady states.
- The systems $\mathsf{Clock}_\mathsf{n} = \begin{pmatrix} {}_{-}{}_{+1} \bmod n \\ \mathrm{id} \end{pmatrix} : \begin{pmatrix} \mathsf{n} \\ \mathsf{n} \end{pmatrix} \leftrightarrows \begin{pmatrix} \{*\} \\ \mathsf{n} \end{pmatrix}$ represents periodic orbits with

periodic parameters whose period divides n.

Note that there is always a particularly simple behavior on a system: the identity behaviors $\mathsf{id} : \mathsf{State}_{\mathsf{T}} \to \mathsf{State}_{\mathsf{T}}$. This says that every system behaves as itself. In particular, Time has a trajectory behavior given by $\mathsf{id} : \mathsf{Time} \to \mathsf{Time}$ (namely, the trajectory $s_t = t$), and Fix has a steady state behavior given by $\mathsf{id} : \mathsf{Fix} \to \mathsf{Fix}$ (namely, the steady state $*$), etc. We refer to the identity behavior of T as the *generic* behavior of type T.

Exercise 2.31. Find a representative system for the following kinds of behavior.

1. An eventually periodic orbit (see Definition 2.21) that takes n steps to get to a period of size m.
2. A steady-looking trajectory (see Definition 2.12).
3. A periodic orbit of period at most n whose parameters aren't necessarily also periodic (see Definition 2.14).
4. An trajectory which yields the same output value at every 10^{th} step. ◊

Remark 2.32. As Exercise 2.31 shows, the difference between a periodic orbit and a periodic orbit with periodic parameters can be surmised precisely by noting that they are represented by systems with different interfaces. The dynamics of the systems are the same, but the interfaces (and accordingly, the exposed variable) are different; this explains how the difference between a periodic orbit and a periodic orbit with periodic parameters is all in the chart.

Exercise 2.33. What kind of behaviors do the following systems represent? First, figure out what kind of charts they have, and then see what a behavior with a given chart is. Describe in your own words.

1. The system Plus with:
   - $\mathsf{State}_{\mathsf{Plus}} = \mathbb{N}$.
   - $\mathsf{Out}_{\mathsf{Plus}} = \mathbb{N}$.
   - $\mathsf{In}_{\mathsf{Plus}} = \mathbb{N}$.
   - $\mathsf{expose}_{\mathsf{Plus}} = \mathrm{id}$.
   - $\mathsf{update}_{\mathsf{Plus}}(t, j) = t + j$.
2. The system T_n with:
   - $\mathsf{State}_{\mathsf{T}_n} = \mathbb{N}$.
   - $\mathsf{Out}_{\mathsf{T}_n} = \{0, \ldots, n-1\}$.
   - $\mathsf{In}_{\mathsf{T}_n} = \{*\}$.
   - $\mathsf{expose}_{\mathsf{T}_n}(t) = t \mod n$.
   - $\mathsf{update}_{\mathsf{T}_n}(t, *) = t + 1$.
3. The system XOR with:
   - $\mathsf{State}_{\mathsf{XOR}} = \mathsf{Bool} = \{\texttt{true}, \texttt{false}\}$.
   - $\mathsf{Out}_{\mathsf{XOR}} = \mathsf{Bool}$.

- $\mathsf{In}_{\mathsf{XOR}} = \mathsf{Bool}$.
- $\mathrm{expose}_{\mathsf{XOR}} = \mathrm{id}$.
- $$\begin{aligned} \mathrm{update}_{\mathsf{XOR}}(\mathtt{true},\mathtt{true}) &= \mathtt{false}, \\ \mathrm{update}_{\mathsf{XOR}}(\mathtt{false},\mathtt{true}) &= \mathtt{true}, \\ \mathrm{update}_{\mathsf{XOR}}(\mathtt{true},\mathtt{false}) &= \mathtt{true}, \\ \mathrm{update}_{\mathsf{XOR}}(\mathtt{false},\mathtt{false}) &= \mathtt{false}. \end{aligned}$$

4. The system List_C for a set of *choices* C with:
   - $\mathsf{State}_{\mathsf{List}_C} = \mathsf{List}_C$ is the set of lists of elements in C.
   - $\mathsf{Out}_{\mathsf{List}_C} = \mathsf{List}_C$.
   - $\mathsf{In}_{\mathsf{List}_C} = C$.
   - $\mathrm{expose}_{\mathsf{List}_C} = \mathrm{id}$.
   - $\mathrm{update}_{\mathsf{List}_C}(\ell, c) = c :: \ell$, that is, we update a list by appending the character $c \in C$ to the start.

◊

While every system T represents some kind of behavior — just take the kind of behavior to be exactly described by behaviors $\mathsf{T} \to \mathsf{S}$ — we are most interested in those simple systems T whose behavior we can fully understand.

We have written a behavior of shape T in S with an arrow $\phi : \mathsf{T} \to \mathsf{S}$. This suggests that there is a category with deterministic systems as its objects and behaviors as its morphisms; and there is!

Definition 2.34. The category $\mathbf{Chart}_{\mathcal{C}}$ of charts in $\mathcal{C}$ has

- Objects the *arenas* $\binom{A^-}{A^+}$, pairs of objects in $\mathcal{C}$.
- Maps the *charts* $\binom{f_\flat}{f} : \binom{A^-}{A^+} \rightrightarrows \binom{B^-}{B^+}$.
- Composition the composite of a chart $\binom{f_\flat}{f} : \binom{A^-}{A^+} \rightrightarrows \binom{B^-}{B^+}$ with a chart $\binom{g_\flat}{g} : \binom{B^-}{B^+} \rightrightarrows \binom{C^-}{C^+}$ is
$$\binom{g_\flat}{g} \circ \binom{f_\flat}{f} := \binom{(a^+, a^-) \mapsto g_\flat(f(a^+), f_\flat(a^+, a^-))}{g \circ f}.$$
- The identity chart is $\binom{\pi_2}{\mathrm{id}} : \binom{A^-}{A^+} \rightrightarrows \binom{A^-}{A^+}$.

Exercise 2.35. Check that $\mathbf{Chart}_{\mathcal{C}}$ is indeed a category. That is,

1. For charts $\binom{f_\flat}{f} : \binom{A^-}{A^+} \rightrightarrows \binom{B^-}{B^+}$, $\binom{g_\flat}{g} : \binom{B^-}{B^+} \rightrightarrows \binom{C^-}{C^+}$, and $\binom{h_\flat}{h} : \binom{C^-}{C^+} \rightrightarrows \binom{D^-}{D^+}$,

show that

$$\begin{pmatrix} h_\flat \\ h \end{pmatrix} \circ \left(\begin{pmatrix} g_\flat \\ g \end{pmatrix} \circ \begin{pmatrix} f_\flat \\ f \end{pmatrix} \right) = \left(\left(\begin{pmatrix} h_\flat \\ h \end{pmatrix} \circ \begin{pmatrix} g_\flat \\ g \end{pmatrix} \right) \circ \begin{pmatrix} f_\flat \\ f \end{pmatrix} \right).$$

2. For a chart $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} A^- \\ A^+ \end{pmatrix} \rightrightarrows \begin{pmatrix} B^- \\ B^+ \end{pmatrix}$, show that

$$\begin{pmatrix} \pi_2 \\ \mathrm{id} \end{pmatrix} \circ \begin{pmatrix} f_\flat \\ f \end{pmatrix} = \begin{pmatrix} f_\flat \\ f \end{pmatrix} = \begin{pmatrix} f_\flat \\ f \end{pmatrix} \circ \begin{pmatrix} \pi_2 \\ \mathrm{id} \end{pmatrix}.$$

◊

Exercise 2.36. What are the charts of the following forms in simpler terms?

1. $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} 1 \\ 1 \end{pmatrix} \rightrightarrows \begin{pmatrix} A^- \\ A^+ \end{pmatrix}$.
2. $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} A^- \\ A^+ \end{pmatrix} \rightrightarrows \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.
3. $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} 1 \\ A^+ \end{pmatrix} \rightrightarrows \begin{pmatrix} B^- \\ B^+ \end{pmatrix}$.

◊

Proposition 2.37. There is a category **Sys** with deterministic systems as its objects and where a map $\mathsf{T} \to \mathsf{S}$ is a pair consisting of a chart $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} \mathsf{In}_\mathsf{T} \\ \mathsf{Out}_\mathsf{T} \end{pmatrix} \rightrightarrows \begin{pmatrix} \mathsf{In}_\mathsf{S} \\ \mathsf{Out}_\mathsf{S} \end{pmatrix}$ and a $\begin{pmatrix} f_\flat \\ f \end{pmatrix}$-behavior $\phi : \mathsf{T} \to \mathsf{S}$. Composition is given by composing both the charts and the functions on states, and identities are given by the generic behaviors: the identity chart with the identity function $\mathrm{id} : \mathsf{State}_\mathsf{T} \to \mathsf{State}_\mathsf{T}$.

Proof. We just need to check that the composite $\psi \circ \phi$ of two behaviors $\phi : \mathsf{T} \to \mathsf{S}$ and $\psi : \mathsf{S} \to \mathsf{U}$ with charts $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} \mathsf{In}_\mathsf{T} \\ \mathsf{Out}_\mathsf{T} \end{pmatrix} \rightrightarrows \begin{pmatrix} \mathsf{In}_\mathsf{S} \\ \mathsf{Out}_\mathsf{S} \end{pmatrix}$ and $\begin{pmatrix} g_\flat \\ g \end{pmatrix} : \begin{pmatrix} \mathsf{In}_\mathsf{S} \\ \mathsf{Out}_\mathsf{S} \end{pmatrix} \rightrightarrows \begin{pmatrix} \mathsf{In}_\mathsf{U} \\ \mathsf{Out}_\mathsf{U} \end{pmatrix}$ is a behavior with chart $\begin{pmatrix} g_\flat \\ g \end{pmatrix} \circ \begin{pmatrix} f_\flat \\ f \end{pmatrix}$. That is, we need to check that Eq. (2.25) is satisfied for $\psi \circ \phi$. We can do this using the fact that it is satisfied for both ψ and ϕ.

$$\begin{aligned} \mathrm{expose}_\mathsf{U}(\psi(\phi(t))) &= \psi(\mathrm{expose}_\mathsf{S}(\phi(t))) \\ &= \psi(\phi(\mathrm{expose}_\mathsf{T}(t))). \end{aligned}$$

$$\begin{aligned} &\mathrm{update}_\mathsf{U}(\psi(\phi(t)), g_\flat(f(\mathrm{expose}_\mathsf{T}(t)), f_\flat(\mathrm{expose}_\mathsf{T}(t), i))) \\ &\qquad = \mathrm{update}_\mathsf{U}(\psi(\phi(t)), g_\flat(\mathrm{expose}_\mathsf{S}(\phi(t)), f_\flat(\mathrm{expose}_\mathsf{T}(t), i))) \\ &\qquad = \psi(\mathrm{update}_\mathsf{S}(\phi(t), f_\flat(\mathrm{expose}_\mathsf{T}(t), i))) \\ &\qquad = \psi(\phi(\mathrm{update}_\mathsf{T}(t, i))). \end{aligned}$$

□

There are two different ways to understand what composition of behaviors means: one based on post-composition, and the other based on pre-composition.

- We see that any behavior $\mathsf{S} \to \mathsf{U}$ gives a way of turning T-shaped behaviors in S to T-shaped behaviors in U.
- We see that any behavior $\mathsf{T} \to \mathsf{S}$ gives a way of turning S-shaped behaviors in U into T-shaped behaviors in U.

Example 2.38. Any steady state s can be seen as a particularly simple trajectory: $s_t = s$ for all t. We have seen in **??** that steady states are Fix-shaped behaviors. We can use composition of behaviors to understand how steady states give rise to trajectories.

The generic steady state $*$ of Fix (that is, the identity behavior of Fix) generates a trajectory $s : \mathbb{N} \to \mathsf{State}_{\mathsf{Fix}}$ with input parameters $p_t = *$ and $s_t = *$. This gives us a behavior $s : \mathsf{Time} \to \mathsf{Fix}$.

Now, for every steady state $\gamma : \mathsf{Fix} \to \mathsf{S}$, we may compose to get a trajectory $\gamma \circ s : \mathsf{Time} \to \mathsf{S}$.

Exercise 2.39. Adapt the argument of Example 2.38 to show that

1. Any eventually periodic orbit gives rise to a trajectory.
2. If n divides m, then any orbit of period at most n gives rise to an orbit of period of most m. ◊

Isomorphisms of Systems Now that we have a category of systems and behaviors, category theory supplies us with a definition of isomorphism for systems.

Definition 2.40. An *isomorphism* of a system T with a system S is a a behavior $\phi : \mathsf{T} \to \mathsf{S}$ for which there is another behavior $\phi^{-1} : \mathsf{S} \to \mathsf{T}$ such that $\phi \circ \phi^{-1} = \mathrm{id}_{\mathsf{S}}$ and $\phi^{-1} \circ \phi = \mathrm{id}_{\mathsf{T}}$.

Let's see that this is indeed a good notion of sameness for systems.

Proposition 2.41. A behavior $\phi : \mathsf{T} \to \mathsf{S}$ is an isomorphism if and only if the following conditions hold:

1. The map $\phi : \mathsf{State}_{\mathsf{T}} \to \mathsf{State}_{\mathsf{S}}$ is an isomorphism of sets — a bijection.
2. The chart $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} \mathsf{In}_{\mathsf{T}} \\ \mathsf{Out}_{\mathsf{T}} \end{pmatrix} \rightrightarrows \begin{pmatrix} \mathsf{In}_{\mathsf{S}} \\ \mathsf{Out}_{\mathsf{S}} \end{pmatrix}$ of ϕ is an isomorphism in $\mathbf{Chart}_{\mathbf{Set}}$. That is, $f : \mathsf{Out}_{\mathsf{T}} \to \mathsf{Out}_{\mathsf{S}}$ is a bijection and there is a bijection $f'_\flat : \mathsf{In}_{\mathsf{T}} \to \mathsf{Out}_{\mathsf{T}}$ such that $f_\flat = f'_\flat \circ \pi_2$.

Proof. Since composition in the category of systems and behaviors is given by composition of the underlying charts and maps, ϕ is an isomorphism of systems if and only if its action on states is a bijection and its chart is an isomorphism in the category of charts. It just remains to see that our description of isomorphism of charts is accurate, which we leave to Exercise 2.42. □

Exercise 2.42. Show that a chart $\binom{f_\flat}{f} : \binom{A^-}{A^+} \rightrightarrows \binom{B^-}{B^+}$ is an isomorphism if and only if f is an isomorphisms and there is an isomorphism $f'_\flat : A^- \to B^-$ such that $f_\flat = f'_\flat \circ \pi_2$. ◊

2.5.1 Internal behaviors

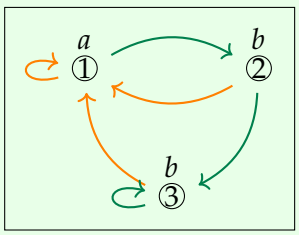
While we will often be interested in behaviors of systems that change the interface in the sense of having non-trivial charts, we will also be interested in behaviors of systems that do not changed the exposed variables at all. These behaviors play a very different role in the theory of dynamical systems than behaviors like trajectories and steady states. Because they don't change observable behavior (since they have identity chart), they say more about how we *model* the observable behavior than what that behavior is itself.

Definition 2.43. Let $\binom{I}{O}$ be an arena. The category

$$\mathbf{Sys}\binom{I}{O}$$

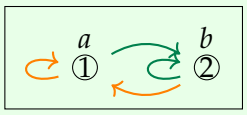
of deterministic $\binom{I}{O}$-systems has as objects the systems $\binom{\text{update}_\mathsf{S}}{\text{expose}_\mathsf{S}} : \binom{\text{State}_\mathsf{S}}{\text{State}_\mathsf{S}} \leftrightarrows \binom{I}{O}$ with interface $\binom{I}{O}$ and as maps the *internal behaviors* $\phi : \mathsf{T} \to \mathsf{S}$, those behaviors whose chart is the identity chart on $\binom{I}{O}$.

Example 2.44. Recall the $\binom{\{\texttt{green},\texttt{orange}\}}{\{a,b\}}$-system S from Example 1.28:



If we had built this system as a model of some relationships between input colors and output letters we were seeing in the wild, then we have made this system a bit redundant. If the output is a, and we feed it `green`, the output will be b; if we feed it `orange`, the output will be a. Similarly, if the output is b — no matter which of states 2 or 3 the system is actually in — and we feed it `green`, the output will again be b, and if we feed it `orange`, the output will be a. And there really isn't much else going on in the system.

We can package this observation into a behavior in $\mathbf{Sys}\left(\begin{smallmatrix}\{\texttt{green},\texttt{orange}\}\\ \{a,b\}\end{smallmatrix}\right)$. Let U be the system



We can give a behavior $q : \mathsf{S} \to \mathsf{U}$ with identity chart as follows defined by

$$q(1) = 1$$
$$q(2) = 2$$
$$q(3) = 2$$

We can check, by cases, that this is indeed a behavior. That it is a behavior in $\mathbf{Sys}\left(\begin{smallmatrix}\{\texttt{green},\texttt{orange}\}\\ \{a,b\}\end{smallmatrix}\right)$ means that it doesn't change the observable behavior.

Example 2.44 also gives us an example of an important relation between systems: *bisimulation*. We saw what it means for two systems to be isomorphic: it means they have isomorphic states and the same dynamics and output relative to those isomorphisms. But this is sometimes too strong a notion of sameness for systems; we want to know when two systems *look the same* on the outside.

Let's see what this notion looks like for deterministic systems; then we will describe it in a doctrinal way.

Definition 2.45. In the deterministic doctrine, a *bisimulation* $\sim$ between $\left(\begin{smallmatrix}I\\ O\end{smallmatrix}\right)$-systems S and U is a relation $\sim : \mathsf{State}_\mathsf{S} \times \mathsf{State}_\mathsf{U} \to \{\texttt{true},\texttt{false}\}$ between states of these systems such that $s \sim u$ only when s and u have related dynamics:

$$s \sim u \text{ implies } \mathrm{expose}_\mathsf{S}(s) = \mathrm{expose}_\mathsf{U}(u)$$
$$s \sim u \text{ implies } \mathrm{update}_\mathsf{S}(s,i) \sim \mathrm{update}_\mathsf{U}(u,i) \text{ for all } i \in I.$$

If $\sim$ is a bisimulation, we say that s and u are *bisimilar* when $s \sim u$.

A bisimulation $\sim$ is said to be *total* if every $s \in \mathsf{State}_\mathsf{S}$ is bisimilar to some $u \in \mathsf{State}_\mathsf{U}$ and vice-versa.

Bisimilarity is a strong relation between states of systems. For deterministic systems, this implies that they act the same on any input.

Proposition 2.46. Let S and U be deterministic $\left(\begin{smallmatrix}I\\ O\end{smallmatrix}\right)$-systems, and let $\sim$ be a total bisimulation between them. If $s_0 \sim u_0$ are bisimilar, then they induce the same transformation on streams of inputs into streams of outputs:

$$\mathrm{transform}^{s_0}_\mathsf{S} = \mathrm{transform}^{u_0}_\mathsf{U}.$$

Proof. Let $i : \mathbb{N} \to I$ be a stream of inputs. Let $s : \mathbb{N} \to \mathsf{State}_\mathsf{S}$ be the stream of states generated by s_0 and similarly, let $u : \mathbb{N} \to \mathsf{State}_\mathsf{U}$ be the stream of states generated by u_0.

We first show that $s_n \sim u_n$ for all n. Our base case holds by hypothesis; now suppose that $s_n \sim u_n$ seeking $s_{n+1} \sim u_{n+1}$. Well,

$$s_{n+1} = \mathrm{update}_\mathsf{S}(s_n, i_n) \sim \mathrm{update}_\mathsf{U}(u_n, i_n) = u_{n+1}$$

because $\sim$ is a bisimulation.

Finally,

$$\mathrm{transforms}_\mathsf{S}(i)_n = \mathrm{expose}_\mathsf{S}(s_n) = \mathrm{expose}_\mathsf{U}(u_n) = \mathrm{transform}_\mathsf{U}(i)_n$$

because $s_n \sim u_n$. □

We can talk about bisimilar states without reference to the particular bisimulation between the systems they are a part of because, as it turns out, being bisimilar is independent of the particular bisimulation. To see this, we need to introduce an interesting system: the system of trees.

Definition 2.47. Let $\binom{I}{O}$ be an arena in the deterministic doctrine. An $\binom{I}{O}$-tree τ (or a O-labeled, I-branching tree) consists of:

- A *root* $\mathrm{root}(\tau) \in O$.
- For each parameter $i \in I$, a *child* tree $\mathrm{child}(\tau, i)$.

Definition 2.48. Let $\binom{I}{O}$ be an arena in the deterministic doctrine. The $\binom{I}{O}$-system $\mathsf{Tree}_{\binom{I}{O}}$ of $\binom{I}{O}$-trees has

- $\mathsf{State}_\mathsf{Tree}$ is the set of $\binom{I}{O}$-trees.
- Each tree exposes its root: $\mathrm{expose}_\mathsf{Tree}(\tau) = \mathrm{root}(\tau)$.
- The system updates by following a tree down the i^{th} branch: $\mathrm{update}_\mathsf{Tree}(\tau, i) = \mathrm{child}(\tau, i)$

We can think of an $\binom{I}{O}$-tree as a stream of possible outputs of an $\binom{I}{O}$-system. In the current state, we see the root of the tree. When we transition to the next state with parameter i, we will see the rest of the output. This observation suggests a universal characterization of the system of $\binom{I}{O}$-trees.

Proposition 2.49. The $\binom{I}{O}$-system $\mathsf{Tree}_{\binom{I}{O}}$ of $\binom{I}{O}$-trees is terminal in the category of $\binom{I}{O}$-systems.

Proof. We will show that there is a unique simulation $!_{\mathsf{S}} : \mathsf{S} \to \mathsf{Tree}_{\binom{I}{O}}$ for any $\binom{I}{O}$-system S. For any $s \in \mathsf{State}_{\mathsf{S}}$, we will define a tree $!_{\mathsf{S}}(s)$ of outputs visible from the state s. We define this as follows:

- The root of $!_{\mathsf{S}}(s)$ is the variable exposed by S:

$$\mathrm{root}(!_{\mathsf{S}}(s)) = \mathrm{expose}_{\mathsf{S}}(s).$$

- The i^{th} child of $!_{\mathsf{S}}(s)$ is the tree of outputs visible from the next state $\mathrm{update}_{\mathsf{S}}(s, i)$:

$$\mathrm{child}(!_{\mathsf{S}}(s), i) = !_{\mathsf{S}}(\mathrm{update}_{\mathsf{S}}(s, i)).$$

Now, we can show that this is a simulation *and* that it is the unique such similation by noticing that this definition is precisely what is required to satisfy the defining laws of a simulation. □

Now we can express the idea that bisimilarity of states is independent of any particular bisimulation between their systems with the following theorem.

Theorem 2.50. Let S and U be $\binom{I}{O}$-systems. A state $s \in \mathsf{State}_{\mathsf{S}}$ is bisimilar to a state $u \in \mathsf{State}_{\mathsf{U}}$ for some bisimulation $\sim$ between S and U if and only if $!_{\mathsf{S}}(s) = !_{\mathsf{U}}(u)$.

Proof. First, let's show that if s is bisimilar to u via some bisimulation $\sim$, then $!_{\mathsf{S}}(s) = !_{\mathsf{U}}(u)$. Now, to show that two trees are equal, we need to show that they have equal roots and equal children.

- The root of $!_{\mathsf{S}}(s)$ is $\mathrm{expose}_{\mathsf{S}}(s)$, and the root of $!_{\mathsf{U}}(u)$ is $\mathrm{expose}_{\mathsf{U}}(u)$. But since $s \sim u$ by hypothesis, these are equal.
- Similarly, the i^{th} child of $!_{\mathsf{S}}(s)$ is $!_{\mathsf{S}}(\mathrm{update}_{\mathsf{S}}(s, i))$, while the i^{th} child of $!_{\mathsf{U}}(u)$ is $!_{\mathsf{U}}(\mathrm{update}_{\mathsf{U}}(u, i))$. But since $\sim$ is a bisimulation, we have that $\mathrm{update}_{\mathsf{S}}(s, i) \sim \mathrm{update}_{\mathsf{U}}(u, i)$, and so by the same argument we are giving, we will find that $!_{\mathsf{S}}(\mathrm{update}_{\mathsf{S}}(s, i)) = !_{\mathsf{U}}(\mathrm{update}_{\mathsf{U}}(u, i))$.[1]

On the other hand, suppose that $!_{\mathsf{S}}(s) = !_{\mathsf{U}}(u)$. We now need to defined a bisimulation $\sim$ between S and U for which $s \sim u$. For any sequence of inputs $i : \mathsf{n} \to I$, we can evolve a system in state s by the entire sequence i to yield a state $\mathrm{update}^*_{\mathsf{S}}(s, i)$ in the following way:

- If $n = 0$, then $\mathrm{update}^*_{\mathsf{S}}(s, i) = s$.
- For $n + 1$, then $\mathrm{update}^*_{\mathsf{S}}(s, i) = \mathrm{update}_{\mathsf{S}}(\mathrm{update}^*_{\mathsf{S}}(s, i|_{\mathsf{n}}), i_{n+1})$.

We may then define $\sim$ in the following way.

$$x \sim y \text{ if and only if there is an } n \in \mathbb{N} \text{ and } i : \mathsf{n} \to I \text{ with } x = \mathrm{update}^*_{\mathsf{S}}(s, i) \text{ and } y = \mathrm{update}^*_{\mathsf{U}}(u, i).$$

[1] This style of proof is called *proof by co-induction*. Where induction assumes a base case and then breaks apart the next step into a smaller step, co-induction shows that the proof can always be continued in a manner which covers all possible options.

It remains to show that this is a bisimulation.

For any $\binom{I}{O}$-tree τ and any n-length sequence $i : \mathsf{n} \to I$ of parameter (for any $n \in \mathbb{N}$), we can follow the path i through the tree τ to get a new tree subtree(τ, i):

- If $n = 0$, then subtree$(\tau, i) = \tau$.
- For $n + 1$, subtree(τ, i) = child(subtree$(\tau, i|_{\mathsf{n}})$) is the i^{th} child of the tree found by following i for the first n steps.

Note that $\mathsf{s}(\text{update}^*_{\mathsf{S}}(s, i)) = \text{subtree}(!_{\mathsf{S}}(s), i)$ by a quick inductive argument. Now we can show that $\sim$ is a bisimulation.

- Suppose that $x \sim y$, seeking to show that $\text{expose}_{\mathsf{S}}(x) = \text{expose}_{\mathsf{U}}(y)$. By hypothesis, $x = \text{update}^*_{\mathsf{S}}(s, i)$ and $y = \text{update}^*_{\mathsf{U}}(u, i)$. But then

$$\begin{aligned}\text{expose}_{\mathsf{S}}(x) &= \text{root}(!_{\mathsf{S}}(x)) \\ &= \text{root}(\text{subtree}(!_{\mathsf{S}}(s), i)) \\ &= \text{root}(\text{subtree}(!_{\mathsf{U}}(u), i)) \\ &= \text{root}(!_{\mathsf{U}}(y)) \\ &= \text{expose}_{\mathsf{U}}(y).\end{aligned}$$

- Suppose that $x \sim y$, seeking to show that $\text{update}_{\mathsf{S}}(x, j) \sim \text{update}_{\mathsf{U}}(y, j)$. By hypothesis, $x = \text{update}^*_{\mathsf{S}}(s, i)$ and same for $y = \text{update}^*_{\mathsf{U}}(u, i)$. Then letting $i' : \mathsf{n}+1 \to \mathbb{N}$ be defined by $i'_{n+1} = j$ and $i'_k = i_k$ otherwise, we see that $\text{update}_{\mathsf{S}}(x, j) = \text{update}^*_{\mathsf{S}}(s, i')$ and $\text{update}_{\mathsf{U}}(y, j) = \text{update}^*_{\mathsf{U}}(y, i')$, so that by definition they are related by $\sim$.

□

2.6 Dealing with two kinds of composition: Double categories

In this section, we will introduce the notion of *double category* to help us deal with our two kinds of composition: the composition of systems, and the composition of behaviors.

Definition 2.51. A *double category* $\mathcal{D}$ has:

- A class $\mathtt{ob}\mathcal{D}$ of *objects*.
- A *horizontal* category $h\mathcal{D}$ whose objects are those of $\mathcal{D}$. We call the maps in $h\mathcal{D}$ the *horizontal* maps of $\mathcal{D}$.
- A *vertical* category $v\mathcal{D}$ whose objects are those of $\mathcal{D}$. We call the maps in $v\mathcal{D}$ the *vertical* maps of $\mathcal{D}$.
- For vertical maps $j : A \to B$ and $k : C \to D$ and horizontal maps $f : A \to C$ and

$g : B \to D$, there is a set of *squares*

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle j}\downarrow & \alpha & \downarrow{\scriptstyle k} \\ C & \xrightarrow[g]{} & D \end{array}$$

- Squares can be composed both horizontally and vertically:

$$\begin{array}{ccccc} A_1 & \xrightarrow{f_1} & A_2 & \xrightarrow{f_2} & A_3 \\ {\scriptstyle j}\downarrow & \alpha & \downarrow{\scriptstyle k} & \beta & \downarrow{\scriptstyle \ell} \\ B_1 & \xrightarrow[g_1]{} & B_2 & \xrightarrow[f_2]{} & B_3 \end{array} \quad \mapsto \quad \begin{array}{ccc} A_1 & \xrightarrow{f_2 f_1} & A_3 \\ {\scriptstyle j}\downarrow & \alpha \mid \beta & \downarrow{\scriptstyle \ell} \\ B_1 & \xrightarrow[g_2 g_1]{} & B_3 \end{array}$$

$$\begin{array}{ccc} A_1 & \xrightarrow{f} & A_2 \\ {\scriptstyle j_1}\downarrow & \alpha & \downarrow{\scriptstyle k_1} \\ B_1 & \xrightarrow{g} & B_2 \\ {\scriptstyle j_1}\downarrow & \beta & \downarrow{\scriptstyle k_2} \\ C_1 & \xrightarrow[h]{} & C_2 \end{array} \quad \mapsto \quad \begin{array}{ccc} A_1 & \xrightarrow{f} & A_3 \\ {\scriptstyle j_2 j_1}\downarrow & \frac{\alpha}{\beta} & \downarrow{\scriptstyle k_2 k_1} \\ C_1 & \xrightarrow[h]{} & B_3 \end{array}$$

- For every vertical map $j : A \to B$, there is an identity square

$$\begin{array}{ccc} A & = & A \\ {\scriptstyle j}\downarrow & j & \downarrow{\scriptstyle j} \\ B & = & B \end{array}$$

which we will also refer to as j, for convenience. Similarly, for every horizontal map $f : A \to B$, there is an identity square

$$\begin{array}{ccc} A & \xrightarrow{f} & A \\ \| & f & \| \\ B & \xrightarrow[f]{} & B \end{array}$$

which we will also refer to as f, for convenience.

- Vertical and horizontal composition is associative and unital, and the *interchange law* holds. That is:
  - For horizontally composable squares α, β, and γ,
  $$(\alpha \mid \beta) \mid \gamma = \alpha \mid (\beta \mid \gamma).$$
  - For vertically composable squares α, β, and γ,[a]
  $$\frac{\left(\frac{\alpha}{\beta}\right)}{\gamma} = \frac{\alpha}{\left(\frac{\beta}{\gamma}\right)}$$
  - For a square α with left and right vertical edges j and k respectively,
  $$j \mid \alpha = \alpha = \alpha \mid k.$$
  - For a square α with top and bottom horizontal edges f and g,
  $$\frac{f}{\alpha} = \alpha = \frac{\alpha}{g}.$$[b]
  - For four appropriately composable squares α, β, γ, and δ, the following interchange law holds:
  $$\frac{\alpha \mid \beta}{\gamma \mid \delta} = \frac{\alpha}{\beta}\Big|\frac{\gamma}{\delta}.$$

[a] If you're seeing this and feeling worried about fractions, you can put your mind at ease; we promise there will be no fractions. Only squares next to squares.

[b] There aren't any fractions here either.

Phew, that was quite the definition! The reason the definition of a double category is so much more involved than the definition of a category is that there is more than twice the data: there's the vertical category and the horizontal category, but also how they interact through the squares.

Remark 2.52. Just like we notate the identity square on a vertical morphism j by j and the identity square on a horizontal morphism f by f, we will often denote composition of vertical morphisms by $\frac{f}{g}$ and of horizontal morphisms by $j \mid k$. This notation agrees with the composition of their respective identity squares, and will be much more pleasant to look at when writing equations.

Finally, we are ready to meet the double category of arenas. This is where our dynamical systems live, and where they behave.

Definition 2.53. The *double category of arenas* in the deterministic doctrine is a double category which has:

- Its objects are the *arenas*, pairs of sets $\binom{A^-}{A^+}$.

- Its horizontal category is the category of charts.
- Its vertical category is the category of lenses.
- There is a square of the following form

$$\begin{array}{ccc} \begin{pmatrix} A^- \\ A^+ \end{pmatrix} & \overset{\begin{pmatrix} f_\flat \\ f \end{pmatrix}}{\Longrightarrow} & \begin{pmatrix} B^- \\ B^+ \end{pmatrix} \\ \begin{pmatrix} j^\sharp \\ j \end{pmatrix} \downarrow\uparrow & & \downarrow\uparrow \begin{pmatrix} k^\sharp \\ k \end{pmatrix} \\ \begin{pmatrix} C^- \\ C^+ \end{pmatrix} & \underset{\begin{pmatrix} g^\sharp \\ g \end{pmatrix}}{\Longrightarrow} & \begin{pmatrix} D^- \\ D^+ \end{pmatrix} \end{array} \tag{2.54}$$

  if and only if the following equations hold:

$$g(j(a^+)) = k(f(a^+)) \tag{2.55}$$

$$k^\sharp(f(a^+), g_\flat(j(a^+), c^-)) = f_\flat(a^+, j^\sharp(a^+, c^-)) \tag{2.56}$$

  for all $a^+ \in A^+$ and $c^- \in C^-$.

It's not obvious from this definition that we actually get a double category with this definition. It's not even clear that we have defined a way to compose the squares vertically and horizontally.

It turns out we don't need to know anything else to know that we can compose these squares, at least in principle. This is because there is at most one square filling any two charts and two lenses that line up as in Eq. (2.54); to compose these squares just means that if we have two such squares lining up, the defining equations Eq. (2.55) hold also for the appropriate composites. We call double categories with this property *thin*.

Definition 2.57. A double category is *thin* if there is at most one square of any signature.

So long as composition is well defined in a thin double category, the laws of associativity and interchange for square composition come for free; there is at most one square of the appropriate signature, so any two you can write down are already equal. We do still have to show that composition is well defined in this way, which we'll do a bit more generally in **??**

Taking for granted that the double category of arenas is indeed a double category, what does this mean for systems? Well, behaviors are particular squares in the double category of arenas.

Proposition 2.58. Let T and S be dynamical systems. A behavior $\phi : \mathsf{T} \to \mathsf{S}$ is equivalently a square of the following form in the double category of arenas:

$$\begin{array}{ccc} \begin{pmatrix} \mathsf{State}_\mathsf{T} \\ \mathsf{State}_\mathsf{T} \end{pmatrix} & \overset{\begin{pmatrix} \phi\circ\pi_2 \\ \phi \end{pmatrix}}{\rightrightarrows} & \begin{pmatrix} \mathsf{States}_\mathsf{S} \\ \mathsf{States}_\mathsf{S} \end{pmatrix} \\ \begin{pmatrix} \mathrm{update}_\mathsf{T} \\ \mathrm{expose}_\mathsf{T} \end{pmatrix} \downarrow\uparrow & & \downarrow\uparrow \begin{pmatrix} \mathrm{update}_\mathsf{S} \\ \mathrm{expose}_\mathsf{S} \end{pmatrix} \\ \begin{pmatrix} \mathsf{In}_\mathsf{T} \\ \mathsf{Out}_\mathsf{T} \end{pmatrix} & \underset{\begin{pmatrix} f^\sharp \\ f \end{pmatrix}}{\rightrightarrows} & \begin{pmatrix} \mathsf{In}_\mathsf{S} \\ \mathsf{Out}_\mathsf{S} \end{pmatrix} \end{array} \tag{2.59}$$

Proof. This is a simple matter of checking the definitions against eachother. The defining equations of Definition 2.53 specialize to the defining equations of Definition 2.24. □

Remark 2.60. While the definition of double category we gave treated both horizontal and vertical directions the same, we will often want to see a square

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle j}\downarrow & \alpha & \downarrow{\scriptstyle k} \\ C & \xrightarrow[g]{} & D \end{array}$$

as a sort of map $\alpha : j \to k$ from its left to its right side, or a map $\alpha : f \to g$ from its top to its bottom side. For example, the systems themselves are certain lenses (vertical maps), and the behaviors are squares between them. On the other hand, we can also see a square as a way of wiring together charts.

Example 2.61. A square

$$\begin{array}{ccc} \begin{pmatrix} A^- \\ A^+ \end{pmatrix} & \overset{\begin{pmatrix} f_\flat \\ f \end{pmatrix}}{\rightrightarrows} & \begin{pmatrix} B^- \\ B^+ \end{pmatrix} \\ \begin{pmatrix} j^\sharp \\ j \end{pmatrix} \downarrow\uparrow & & \downarrow\uparrow \begin{pmatrix} k^\sharp \\ k \end{pmatrix} \\ \begin{pmatrix} C^- \\ C^+ \end{pmatrix} & \underset{\begin{pmatrix} g^\sharp \\ g \end{pmatrix}}{\rightrightarrows} & \begin{pmatrix} D^- \\ D^+ \end{pmatrix} \end{array} \tag{2.62}$$

can be seen as a chart between lenses, that is, two charts which are compatible according

to the wiring pattern the lenses describe. For example, consider a square of the following form where $\begin{pmatrix} w^\sharp \\ w \end{pmatrix}$ is a wiring diagram:

$$\begin{array}{ccc} \begin{pmatrix} 1 \\ 1 \end{pmatrix} & \xrightarrow[\substack{\longrightarrow}]{\begin{pmatrix} b^- \\ b^+ \end{pmatrix}} & \begin{pmatrix} B^- \\ B^+ \end{pmatrix} \\ \Vert & & \downarrow\uparrow \begin{pmatrix} w^\sharp \\ w \end{pmatrix} \\ \begin{pmatrix} 1 \\ 1 \end{pmatrix} & \xrightarrow[\substack{\longrightarrow \\ \begin{pmatrix} d^- \\ d^+ \end{pmatrix}}]{} & \begin{pmatrix} D^- \\ D^+ \end{pmatrix} \end{array}$$

By Exercise 2.36, we know that the charts in this diagram are pairs of elements $\begin{pmatrix} b^- \\ b^+ \end{pmatrix}$ and $\begin{pmatrix} d^- \\ d^+ \end{pmatrix}$ in the arenas $\begin{pmatrix} B^- \\ B^+ \end{pmatrix}$ and $\begin{pmatrix} D^- \\ D^+ \end{pmatrix}$ respectively. The square then says that $\begin{pmatrix} d^- \\ d^+ \end{pmatrix}$ are the values you would get if you passed $\begin{pmatrix} b^- \\ b^+ \end{pmatrix}$ along the wires in the wiring diagram $\begin{pmatrix} w^\sharp \\ w \end{pmatrix}$.

Let's take a minute to see what composition of squares in the double category of arenas means for systems. Horizontal composition is familiar because it's what lets us compose behaviors:

$$\begin{array}{ccccc} \begin{pmatrix} \mathsf{State}_\mathsf{T} \\ \mathsf{State}_\mathsf{T} \end{pmatrix} & \xrightarrow[\substack{\longrightarrow}]{\begin{pmatrix} \phi\circ\pi_2 \\ \phi \end{pmatrix}} & \begin{pmatrix} \mathsf{State}_\mathsf{S} \\ \mathsf{State}_\mathsf{S} \end{pmatrix} & \xrightarrow[\substack{\longrightarrow}]{\begin{pmatrix} \psi\circ\pi_2 \\ \psi \end{pmatrix}} & \begin{pmatrix} \mathsf{State}_\mathsf{U} \\ \mathsf{State}_\mathsf{U} \end{pmatrix} \\ \begin{pmatrix} \mathrm{update}_\mathsf{T} \\ \mathrm{expose}_\mathsf{T} \end{pmatrix}\downarrow\uparrow & & \downarrow\uparrow\begin{pmatrix} \mathrm{update}_\mathsf{S} \\ \mathrm{expose}_\mathsf{S} \end{pmatrix} & & \downarrow\uparrow\begin{pmatrix} \mathrm{update}_\mathsf{U} \\ \mathrm{expose}_\mathsf{U} \end{pmatrix} \\ \begin{pmatrix} \mathsf{In}_\mathsf{T} \\ \mathsf{Out}_\mathsf{T} \end{pmatrix} & \xrightarrow[\substack{\longrightarrow \\ \begin{pmatrix} f_\flat \\ f \end{pmatrix}}]{} & \begin{pmatrix} \mathsf{In}_\mathsf{S} \\ \mathsf{Out}_\mathsf{S} \end{pmatrix} & \xrightarrow[\substack{\longrightarrow \\ \begin{pmatrix} g_\flat \\ g \end{pmatrix}}]{} & \begin{pmatrix} \mathsf{In}_\mathsf{U} \\ \mathsf{Out}_\mathsf{U} \end{pmatrix} \end{array} = \begin{array}{ccc} \begin{pmatrix} \mathsf{State}_\mathsf{T} \\ \mathsf{State}_\mathsf{T} \end{pmatrix} & \xrightarrow[\substack{\longrightarrow}]{\begin{pmatrix} \psi\phi\circ\pi_2 \\ \psi\phi \end{pmatrix}} & \begin{pmatrix} \mathsf{State}_\mathsf{U} \\ \mathsf{State}_\mathsf{U} \end{pmatrix} \\ \begin{pmatrix} \mathrm{update}_\mathsf{T} \\ \mathrm{expose}_\mathsf{T} \end{pmatrix}\downarrow\uparrow & & \downarrow\uparrow\begin{pmatrix} \mathrm{update}_\mathsf{U} \\ \mathrm{expose}_\mathsf{U} \end{pmatrix} \\ \begin{pmatrix} \mathsf{In}_\mathsf{T} \\ \mathsf{Out}_\mathsf{T} \end{pmatrix} & \xrightarrow[\substack{\longrightarrow \\ \begin{pmatrix} f_\flat \\ f \end{pmatrix} \fatsemi \begin{pmatrix} g_\flat \\ g \end{pmatrix}}]{} & \begin{pmatrix} \mathsf{In}_\mathsf{U} \\ \mathsf{Out}_\mathsf{U} \end{pmatrix} \end{array}$$

On the other hand, vertical composition tells us something else interesting: if you get a chart $\begin{pmatrix} g_\flat \\ g \end{pmatrix}$ by wiring together a chart $\begin{pmatrix} f_\flat \\ f \end{pmatrix}$, then a behavior ϕ with chart $\begin{pmatrix} f_\flat \\ f \end{pmatrix}$

induces a behavior with chart $\begin{pmatrix} g_\flat \\ g \end{pmatrix}$ on the wired together systems.

$$\begin{array}{ccc} \begin{pmatrix} \mathsf{State}_\mathsf{T} \\ \mathsf{State}_\mathsf{T} \end{pmatrix} & \overset{\begin{pmatrix} \phi\circ\pi_2 \\ \phi \end{pmatrix}}{\Longrightarrow} & \begin{pmatrix} \mathsf{States} \\ \mathsf{States} \end{pmatrix} \\ \begin{pmatrix} \mathrm{update}_\mathsf{T} \\ \mathrm{expose}_\mathsf{T} \end{pmatrix}\!\downarrow\!\uparrow & & \downarrow\!\uparrow\!\begin{pmatrix} \mathrm{update}_\mathsf{S} \\ \mathrm{expose}_\mathsf{S} \end{pmatrix} \\ \begin{pmatrix} \mathsf{In}_\mathsf{T} \\ \mathsf{Out}_\mathsf{T} \end{pmatrix} & \underset{\begin{pmatrix} f_\flat \\ f \end{pmatrix}}{\Longrightarrow} & \begin{pmatrix} \mathsf{In}_\mathsf{S} \\ \mathsf{Out}_\mathsf{S} \end{pmatrix} \\ \begin{pmatrix} j^\sharp \\ j \end{pmatrix}\!\downarrow\!\uparrow & & \downarrow\!\uparrow\!\begin{pmatrix} k^\sharp \\ k \end{pmatrix} \\ \begin{pmatrix} I \\ O \end{pmatrix} & \underset{\begin{pmatrix} g_\flat \\ g \end{pmatrix}}{\Longrightarrow} & \begin{pmatrix} I' \\ O' \end{pmatrix} \end{array} = \begin{array}{ccc} \begin{pmatrix} \mathsf{State}_\mathsf{T} \\ \mathsf{State}_\mathsf{T} \end{pmatrix} & \overset{\begin{pmatrix} \phi\circ\pi_2 \\ \psi\phi \end{pmatrix}}{\Longrightarrow} & \begin{pmatrix} \mathsf{States} \\ \mathsf{States} \end{pmatrix} \\ \begin{pmatrix} f^\sharp \\ f \end{pmatrix}\circ\begin{pmatrix} \mathrm{update}_\mathsf{T} \\ \mathrm{expose}_\mathsf{T} \end{pmatrix}\!\downarrow\!\uparrow & & \downarrow\!\uparrow\!\begin{pmatrix} k^\sharp \\ k \end{pmatrix}\circ\begin{pmatrix} \mathrm{update}_\mathsf{S} \\ \mathrm{expose}_\mathsf{S} \end{pmatrix} \\ \begin{pmatrix} I \\ O \end{pmatrix} & \underset{\begin{pmatrix} g_\flat \\ g \end{pmatrix}}{\Longrightarrow} & \begin{pmatrix} I \\ O' \end{pmatrix} \end{array}$$

Example 2.63. Continuing from Example 2.61, suppose that we have a $\begin{pmatrix} b^- \\ b^+ \end{pmatrix}$-steady state s in a system S:

$$\begin{array}{ccc} \begin{pmatrix} 1 \\ 1 \end{pmatrix} & \overset{\begin{pmatrix} s \\ s \end{pmatrix}}{\Longrightarrow} & \begin{pmatrix} \mathsf{States} \\ \mathsf{States} \end{pmatrix} \\ \| & & \downarrow\!\uparrow\!\begin{pmatrix} \mathrm{update}_\mathsf{S} \\ \mathrm{expose}_\mathsf{S} \end{pmatrix} \\ \begin{pmatrix} 1 \\ 1 \end{pmatrix} & \underset{\begin{pmatrix} b^+ \\ b^- \end{pmatrix}}{\Longrightarrow} & \begin{pmatrix} B^- \\ B^+ \end{pmatrix} \end{array} \tag{2.64}$$

We can see that s is a $\begin{pmatrix} d^- \\ d^+ \end{pmatrix}$-steady state of the wired system by vertically composing the square in Eq. (2.64) with the square in Eq. (2.62). This basic fact underlies our arguments in Section 3.2.

We'll return to this idea in Example 3.57.

Recall the categories $\mathbf{Sys}\begin{pmatrix} I \\ O \end{pmatrix}$ of systems with the interface $\begin{pmatrix} I \\ O \end{pmatrix}$ from Definition 2.43. One thing that vertical composition in the double category of arenas shows us is that wiring together systems is functorial with respect to behaviors that don't change the interface.

Proposition 2.65. For a lens $\begin{pmatrix} f^\sharp \\ f \end{pmatrix} : \begin{pmatrix} I \\ O \end{pmatrix} \leftrightarrows \begin{pmatrix} I' \\ O' \end{pmatrix}$, we get a functor

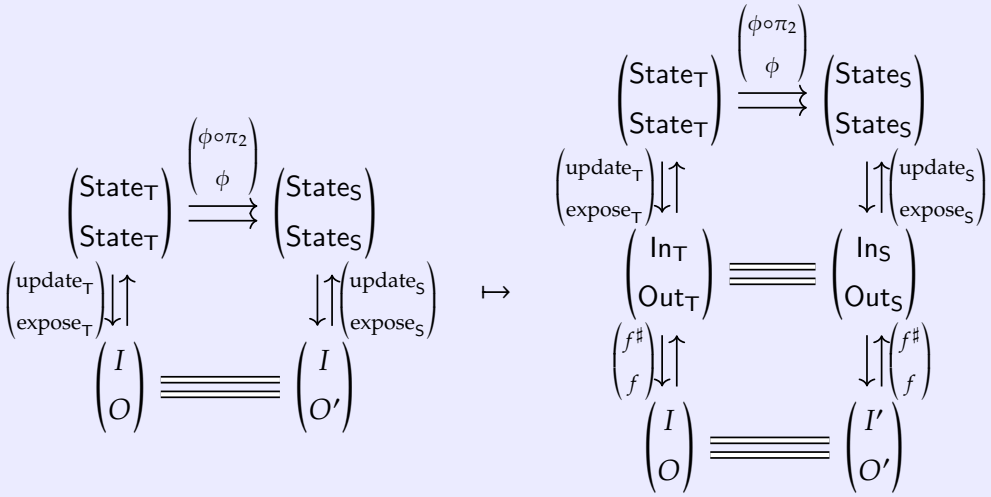
$$\mathbf{Sys}\begin{pmatrix} f^\sharp \\ f \end{pmatrix} : \mathbf{Sys}\begin{pmatrix} I \\ O \end{pmatrix} \to \mathbf{Sys}\begin{pmatrix} I \\ O \end{pmatrix}$$

Given by composing with $\begin{pmatrix} f^\sharp \\ f \end{pmatrix}$:

- For a system $\mathsf{S} = \begin{pmatrix} \mathrm{update}_\mathsf{S} \\ \mathrm{expose}_\mathsf{S} \end{pmatrix} : \begin{pmatrix} \mathsf{State}_\mathsf{S} \\ \mathsf{State}_\mathsf{S} \end{pmatrix} \leftrightarrows \begin{pmatrix} I \\ O \end{pmatrix}$,

$$\mathbf{Sys}\begin{pmatrix} f^\sharp \\ f \end{pmatrix}(\mathsf{S}) = \begin{pmatrix} f^\sharp \\ f \end{pmatrix} \circ \begin{pmatrix} \mathrm{update}_\mathsf{S} \\ \mathrm{expose}_\mathsf{S} \end{pmatrix}.$$

- For a behavior, $\mathbf{Sys}\begin{pmatrix} f^\sharp \\ f \end{pmatrix}$ acts in the following way:



Proof. The functoriality of this construction can be seen immediately from the interchange law of the double category:

$$\frac{\begin{pmatrix} \phi\circ\pi_2 \\ \phi \end{pmatrix} \Big| \begin{pmatrix} psi\circ\pi_2 \\ \psi \end{pmatrix}}{\begin{pmatrix} f^\sharp \\ f \end{pmatrix}} = \frac{\begin{pmatrix} \phi\circ\pi_2 \\ \phi \end{pmatrix} \Big| \begin{pmatrix} psi\circ\pi_2 \\ \psi \end{pmatrix}}{\begin{pmatrix} f^\sharp \\ f \end{pmatrix} \Big| \begin{pmatrix} f^\sharp \\ f \end{pmatrix}} \qquad \text{by the horizontal identity law,}$$

$$= \frac{\begin{pmatrix} \phi\circ\pi_2 \\ \phi \end{pmatrix}}{\begin{pmatrix} f^\sharp \\ f \end{pmatrix}} \Bigg| \frac{\begin{pmatrix} \psi\circ\pi_2 \\ \psi \end{pmatrix}}{\begin{pmatrix} f^\sharp \\ f \end{pmatrix}} \qquad \text{by the interchange law.}$$

□

Chapter 3

Behaviors of the whole from behaviors of the parts

3.1 Introduction

Let's take stock of where we've been so far in the past couple chapters.

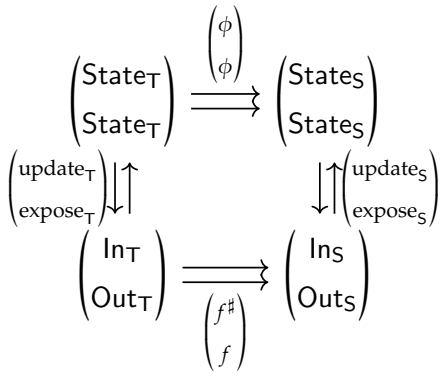
- In Section 1.2.1, we saw the definition of a *deterministic system*.
- In **??**, we learned about *lenses*. We saw how systems can be interpreted as special sorts of lenses, and how we can wire together systems using lens composition.
- In **??**, we learned about behaviors and *charts*. We saw how to define behaviors of systems using the notion of chart. Finally, we saw how the steady states of wired together systems can be calculated from their component systems with matrix arithmetic.

The two sorts of composition we have seen so far — lens composition and chart composition — mirror the two sorts of composition at play in systems theory:

- We can compose *systems* by wiring them together. This uses lens composition.
- We can compose *behaviors* of systems like we compose functions. This uses chart composition.

In this section, we will see how these two sorts of composition interact. Because there are two sorts of composition involved, we will use the notion of a *double category*. A double category is like a category, but there are two sorts of map between the objects, and there is a notion of interaction between these two sorts of map. We'll get to see

that behaviors $\phi : \mathsf{T} \to \mathsf{S}$ of systems (Definition 2.24) are squares



in the double category of *arenas*, Proposition 2.58.

Along the way, we'll see a new interpretation of the categories of lenses and charts which explain why they seem so eerily familiar. We will see them as instances of a very general construction that forms a category out of an *indexed category*: the *Grothendieck construction*. We take this detour into the abstract because when we move to different doctrines in later chapters, the particular notion of lens and chart might change, but they will always be built out of an indexed category in the way we will see in this section.

3.2 Steady states compose according to the laws of matrix arithmetic

We have seen how we can compose systems, and we have seen how systems behave. We have seen a certain composition of behaviors, a form of transitivity that says that if we have a T-shaped behavior in S and a S-shaped behavior in U, then we get a T-shaped behavior in U. But what's the relationship between composing systems and composing their behaviors?

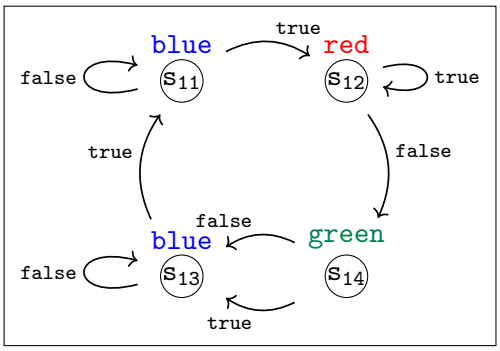
This question will be a major theme of this book. In this section we will give a taste by showing how steady states compose. Later, in **??**, we will see a very abstract theorem that generalizes what we do here for steady states to something that works for *all behaviors*. But in order for that abstract theorem to make sense, we should first see the concrete case of steady states in detail.

Recall that the chart of a steady state $s \in \mathsf{State}_\mathsf{S}$ is the pair $\binom{i}{o}$ with $o = \mathsf{expose}_\mathsf{S}(s)$ and $\mathsf{update}_\mathsf{S}(s, i) = s$. The set of all possible charts for steady states is therefore $\mathsf{In}_\mathsf{S} \times \mathsf{Out}_\mathsf{S}$, and for every chart $\binom{i}{o}$ we have the set $\mathsf{Steady}_{SysS}\binom{i}{o}$ of steady states for this chart.

We can see this function $\mathsf{Steady}_\mathsf{S} : \mathsf{In}_\mathsf{S} \times \mathsf{Out}_\mathsf{S} \to \mathbf{Set}$ as a *matrix of sets* with $\mathsf{Steady}_\mathsf{S}\binom{i}{o}$

in the row i and column o. For example, consider system S_1 of Exercise 1.67:

$\mathsf{S}_1 :=$



(3.1)

This has output value set Colors = {`blue`, `red`, `green`} and input parameter set Bool = {`true`, `false`}. Here is its (Colors × Bool) steady state matrix:

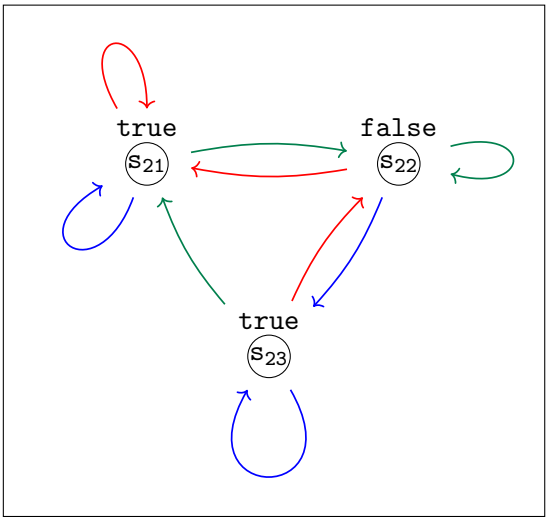
$$\mathsf{Steady}_{\mathsf{S}_1} = \begin{array}{c} \\ \texttt{true} \\ \texttt{false} \end{array} \begin{array}{c} \begin{array}{ccc} \texttt{blue} & & \texttt{red} \end{array} \quad \texttt{green} \\ \begin{bmatrix} \emptyset & \{\texttt{red}\ \circlearrowright_{\texttt{true}}\ (\mathsf{s}_{12})\} & \emptyset \\ \{\texttt{false}\ \circlearrowright\ \texttt{blue}\ (\mathsf{s}_{11}),\ \texttt{false}\ \circlearrowright\ \texttt{blue}\ (\mathsf{s}_{13})\} & \emptyset & \emptyset \end{bmatrix} \end{array} \tag{3.2}$$

If we just want to know how many $\binom{i}{o}$-steady states there are, and not precisely which states they are, we can always take the cardinality of the sets in our matrix of sets to get a bona-fide matrix of numbers. Doing this to the above matrix gives us the matrix

$$\begin{array}{c} \\ \texttt{true} \\ \texttt{false} \end{array} \begin{array}{c} \begin{array}{ccc} \texttt{blue} & \texttt{red} & \texttt{green} \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \end{bmatrix} \end{array}$$

Now, let's take a look at system S_2 from the same exercise:

$\mathsf{S}_2 :=$



This has steady state matrix:

$$\mathsf{Steady}_{\mathsf{S}_2} = \begin{array}{c} \\ \texttt{blue} \\ \texttt{red} \\ \texttt{green} \end{array} \begin{array}{c} \texttt{true} \\ \left[\begin{array}{c} \left\{ \circlearrowright\ \texttt{true}\ (\mathsf{s}_{21}),\ \circlearrowright\ \texttt{true}\ (\mathsf{s}_{23}) \right\} \\ \left\{ \circlearrowright\ \texttt{true}\ (\mathsf{s}_{21}) \right\} \\ \emptyset \end{array}\right. \end{array} \begin{array}{c} \texttt{false} \\ \left.\begin{array}{c} \emptyset \\ \emptyset \\ \left\{ \circlearrowright\ \texttt{false}\ (\mathsf{s}_{22}) \right\} \end{array}\right] \end{array} \tag{3.3}$$

Or, again, if we just want to know how many steady states there are for each chart:

$$\mathsf{Steady}_{\mathsf{S}_2} = \begin{array}{c} \\ \texttt{blue} \\ \texttt{red} \\ \texttt{green} \end{array} \begin{array}{c} \texttt{true} \quad \texttt{false} \\ \begin{bmatrix} 2 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}$$

We can wire these systems together to get a system S:

$$\mathsf{S} := \boxed{\mathsf{S}_1 \quad \mathsf{S}_2}$$

With just a bit of thought, we can find the steady states of this systems without fully calculating its dynamics. A state of S is a pair of states $s_1 \in \mathsf{States}_{\mathsf{S}_1}$ and $s_2 \in \mathsf{States}_{\mathsf{S}_2}$, so for it to be steady both its constituent states must be steady. So let $\binom{i}{o} : \binom{1}{1} \rightrightarrows \binom{\mathsf{Bool}}{\mathsf{Bool}}$ be a chart for S — a pair of booleans. We need s_1 and s_2 to both be steady, so in particular s_1 must be steady at the input i, and s_2 must expose o; but, most importantly, s_2 must then be steady at the input $\mathsf{expose}_{\mathsf{S}_1}(s_1)$ which s_1 exposes.

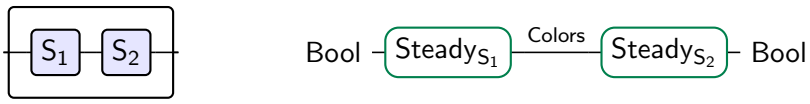
So, to find the set of $\binom{\texttt{true}}{\texttt{true}}$-steady states of S, we must a state of S_1 which is steady for the input `true` and then a steady state of S_2 whose input is what that state outputs and whose output is `true`. There are three pieces of data here: the state of S_1, the state of S_2, and the intermediate value expose by the first state and input into the second state. We can therefore describe the set of $\binom{\texttt{true}}{\texttt{true}}$-steady states of S like this:

$$\begin{aligned} \mathsf{Steady}_{\mathsf{S}}\begin{pmatrix}\texttt{true}\\ \texttt{true}\end{pmatrix} &= \left\{ (m, s_1, s_2) \middle| s_1 \in \mathsf{Steady}_{\mathsf{S}_1}\begin{pmatrix}\texttt{true}\\ m\end{pmatrix}, s_2 \in \mathsf{Steady}_{\mathsf{S}_2}\begin{pmatrix} m \\ \texttt{true}\end{pmatrix} \right\} \\ &= \sum_{m \in \mathsf{Colors}} \mathsf{Steady}_{\mathsf{S}_1}\begin{pmatrix}\texttt{true}\\ m\end{pmatrix} \times \mathsf{Steady}_{\mathsf{S}_2}\begin{pmatrix} m \\ \texttt{true}\end{pmatrix}. \end{aligned}$$

This formula looks very suspiciously like matrix multiplication! Indeed, if we multiply the matrices of numbers of steady states from S_1 and S_2, we get:

$$\begin{matrix} \\ \texttt{true} \\ \texttt{false} \end{matrix}\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \end{bmatrix}\begin{matrix} \texttt{true} & \texttt{false} \\ \end{matrix}\begin{bmatrix} 2 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{matrix} \texttt{true} & \texttt{false} \\ \end{matrix}\begin{matrix} \\ \texttt{true} \\ \texttt{false} \end{matrix}\begin{bmatrix} 1 & 0 \\ 4 & 0 \end{bmatrix}$$

which is the matrix of how many steady states S has! What's even more suspicious is that our wiring diagram for S looks a lot like the string diagram we would use to describe the multiplication of matrices:

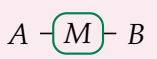


This can't just be a coincidence. Luckily for our sanity, it isn't. In the remainder of this section, we will show how various things one can do with matrices — multiply them, trace them, Kronecker product them — can be done for matrices of sets, and how if your wiring diagram looks like its telling you to that thing, then you can do that thing to the steady states of your internal systems to get the steady states of the whole wired system

Matrices of sets We'll be working with matrices of sets — now and in the coming section — quite a bit, so we should really nail them down. Matrices of sets work a lot like matrices of numbers, especially when the sets are finite; then they are very nearly the same thing as matrices of whole numbers. But the matrix arithmetic of infinite sets works just the same as with finite sets, so we'll do everything in that generality.[1]

Definition 3.4. Let A and B be two sets. $B \times A$ *matrix of sets* is a dependent set $M : B \times A \to \mathbf{Set}$. For $a \in A$ and $b \in B$, we write M_{ba} or $M_{(b,a)}$ for set indexed by a and b, and call this the (b, a)-entry of the matrix M.

We draw of matrix of sets with the following string diagram:



Remark 3.5. We can see a dependent set $X_\_ : A \to \mathbf{Set}$ through the matrix of sets point of view as a *vector of sets*. This is because $X_\_$ is equivalently given by $X_\_ : A \times 1 \to \mathbf{Set}$, which we see is a $A \times 1$ matrix of sets. A $n \times 1$ matrix is equivalently a column vector.

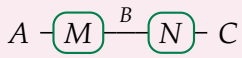
Now we'll go through and define the basic operations of matrix arithmetic: mutliplication, Kronecker product (also known as the tensor product), and partial trace.

[1] This will help us later when we deal with behaviors that have more complicated charts. For example, even finite systems can have infinitely many different trajectories, so we really need the infinite sets.

Definition 3.6. Given an $B \times A$ matrix of sets M and a $C \times B$ matrix of sets N, their *product* NM (or $M \times_B N$ for emphasis) is the $C \times A$ matrix of sets with entries

$$NM_{ca} = \sum_{b \in B} N_{cb} \times M_{ba}.$$

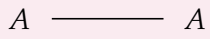
We draw the multiplication of matrices of sets with the following string diagram:



The identity matrix I_A is an $A \times A$ matrix with entries

$$I_{aa'} = \begin{cases} 1 & \text{if } a = a' \\ \emptyset & \text{if } a \neq a' \end{cases}.$$

We draw the identity matrix as a string with no beads on it.



Exercise 3.7. Multiplication of matrices of sets satisfies the usual properties of associativity and unity, but only up to isomorphism. Let M be a $B \times A$ matrix, N a $C \times B$ matrix, and L a $D \times C$ of sets. Show that

1. For all $a \in A$ and $d \in D$, $((LN)M)_{da} \cong (L(NM))_{da}$.
2. For all $a \in A$ and $b \in B$, $(MI_A)_{ba} \cong M_{ba} \cong (I_B M)_{ba}$.

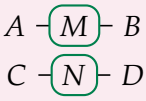
◊

Remark 3.8. The isomorphisms you defined in Exercise 3.7 are *coherent*, much in the way the associativity and unity isomorphisms of a monoidal category are. Together, this means that there is a *bicategory* of sets and matrices of sets between them.

Definition 3.9. Let M be a $B \times A$ matrix and N a $C \times D$ matrix of sets. Their *Kronecker product* or *tensor product* $M \otimes N$ is a $(B \times C) \times (A \times D)$ matrix of sets with entries:

$$(M \otimes N)_{(b,c)(a,d)} = M_{ba} \times N_{cd}.$$

We draw the tensor product $M \otimes M$ of matrices as:

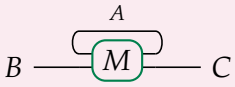


Finally, we need to define the partial trace of a matrix of sets.

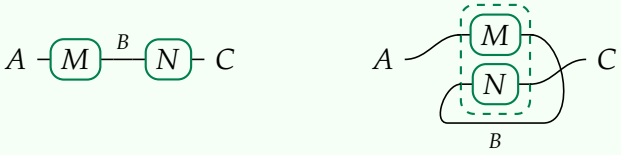
Definition 3.10. Suppose that M is a $(A \times C) \times (A \times B)$ matrix of sets. Its *partial trace* $\mathrm{tr}_A M$ is a $C \times B$ matrix of sets with entries:

$$(\mathrm{tr}_A) M_{cb} = \sum_{a \in A} M_{(a,c)(a,b)}.$$

We draw the partial trace of a matrix of sets as:



Exercise 3.11. Here's an important sanity check we should do about our string diagrams for matrices of sets. The following two diagrams should describe the same matrix, even though they describe it in different ways:



The diagram on the left says "multiply M and N", while the diagram on the right says "tensor M and N, and then partially trace them.". Show that these two diagrams do describe the same matrix:

$$NM \cong \mathrm{tr}_B(M \otimes N).$$

Compare this to Example 1.60, where we say that wiring an input of a system to an output of another can be seen as first taking their parallel product, and then forming a loop. ◊

Steady states and matrix arithmetic For the remainder of this section, we will show that we can calculate the steady state matrix of a wired together system in terms of its component system in a very simple way:

- First, take the steady state matrices of the component systems.
- Then consider the wiring diagram as a string diagram for multiplying, tensoring, and tracing matrices.
- Finally, finish by doing all those operations to the matrix.

In **??**, we will see that this method — or something a lot like it — works calculating the behaviors of a composite system out of the behaviors of its components, as long as the representative of that behavior exposes its entire state. That result will be nicely packaged in a beautiful categorical way: we'll make an *doubly indexed functor*.

But for now, let's just show that tensoring and partially tracing steady state matrices correponds to taking the parallel product and wiring an input to an output, respectively, of systems.

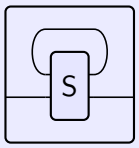
Proposition 3.12. Let S_1 and S_2 be systems. Then the steady state matrix of the parallel product $\mathsf{S}_1 \otimes \mathsf{S}_2$ is the tensor of their steady state matrices:

$$\mathsf{Steady}_{\mathsf{S}_1 \otimes \mathsf{S}_2} \cong \mathsf{Steady}_{\mathsf{S}_1} \otimes \mathsf{Steady}_{\mathsf{S}_2}.$$

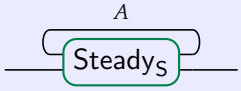
Proof. First, we note that these are both $(\mathsf{Out}_{\mathsf{S}_1} \times \mathsf{Out}_{\mathsf{S}_2}) \times (\mathsf{In}_{\mathsf{S}_1} \times \mathsf{In}_{\mathsf{S}_2})$-matrices of sets. Now, on a chart $\begin{pmatrix} (i_1,i_2) \\ (o_1,o_2) \end{pmatrix}$, a steady state in $\mathsf{S}_1 \otimes \mathsf{S}_2$ will be a pair $(s_1, s_2) \in \mathsf{States}_{\mathsf{S}_1} \times \mathsf{States}_{\mathsf{S}_2}$ such that $\mathsf{update}_{\mathsf{S}_j}(s_j, i_j) = s_j$ and $\mathsf{expose}_{\mathsf{S}_j}(s_j) = o_j$ for $j = 1, 2$. In other words, its just a pair of steady states, one in S_1 and one in S_2. This is precisely the $\begin{pmatrix} (i_1,i_2) \\ (o_1,o_2) \end{pmatrix}$-entry of the right hand side above. □

Remark 3.13. Proposition 3.12 is our motiviation for using the symbol "⊗" for the parallel product of systems.

Proposition 3.14. Let S be a system with $\mathsf{In}_{\mathsf{S}} = A \times B$ and $\mathsf{Out}_{\mathsf{S}} = A \times C$. Let S' be the system formed by wiring the A output into the A input of S:

$\mathsf{S}' :=$ 

Then the steady state matrix of S' is given by partially tracing out A in the steady state matrix of S:

$\mathsf{Steady}_{\mathsf{S}'} =$ 

$= \mathrm{tr}_A\,(\mathsf{Steady}_{\mathsf{S}})$

Proof. Let's first see what a steady state of S' would be. Since S' is just a rewiring of S, it has the same states; so, a steady state s of S' is in particular a state of S. Now,

$$\mathsf{update}_{\mathsf{S}'}(s, b) = \mathsf{update}_{\mathsf{S}}(s, (\pi_1 \mathsf{expose}_{\mathsf{S}}(s), b))$$

by definition, so if $\mathsf{update}_{\mathsf{S}'}(s, b) = s$, then $\mathsf{update}_{\mathsf{S}}(s, (\pi_1 \mathsf{expose}_{\mathsf{S}}(s), b)) = s$. If also $\mathsf{expose}_{\mathsf{S}'}(s) = c$ (so that s is a $\begin{pmatrix} b \\ c \end{pmatrix}$-steady state of S'), then $\pi_2 \mathsf{expose}_{\mathsf{S}}(s) = \mathsf{expose}_{\mathsf{S}'}(s) = c$ as well. In total then, starting with a $\begin{pmatrix} b \\ c \end{pmatrix}$-steady state s of S', we get a $\begin{pmatrix} (\pi_1 \mathsf{expose}_{\mathsf{S}}(s), b) \\ (\pi_1 \mathsf{expose}_{\mathsf{S}}(s), c) \end{pmatrix}$-steady state of S. That is, we have a function

$$s \mapsto (\pi_1 \mathsf{expose}_{\mathsf{S}}(s), s) : \mathsf{Steady}_{\mathsf{S}'} \begin{pmatrix} b \\ c \end{pmatrix} \to (\mathrm{tr}_A \mathsf{Steady}_{\mathsf{S}}) \begin{pmatrix} b \\ c \end{pmatrix}.$$

It remains to show that this function is a bijection. So, suppose we have a pair $(a, s) \in \mathrm{tr}_A \mathsf{Steady}_{\mathsf{S}} \begin{pmatrix} b \\ c \end{pmatrix}$ of an $a \in A$ and a $\begin{pmatrix} (a,b) \\ (a,c) \end{pmatrix}$ steady state of S. Then

$$\mathsf{update}_{\mathsf{S}'}(s, b) = \mathsf{update}_{\mathsf{S}}(s, (\pi_1 \mathsf{expose}_{\mathsf{S}}(s), b))$$

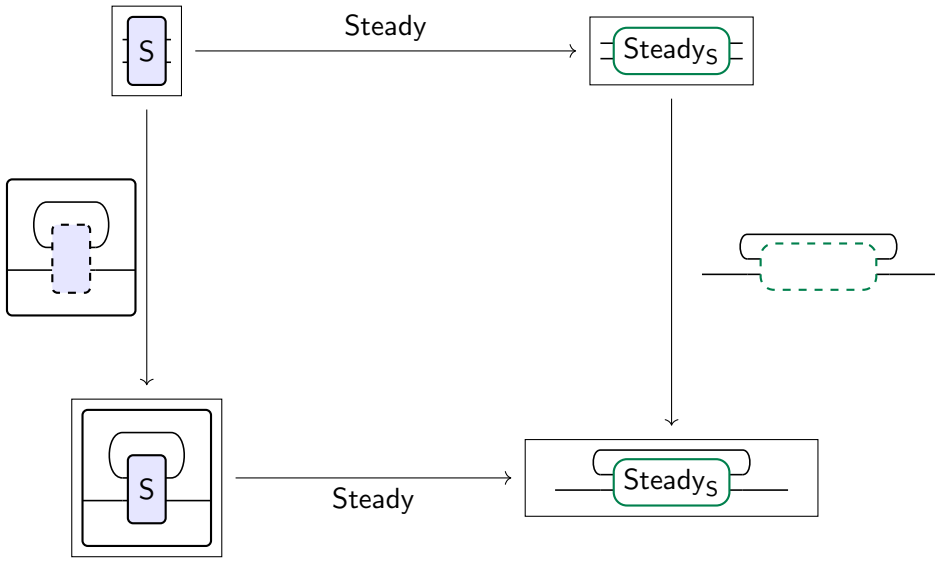
$$= \text{update}_{\mathsf{S}}(s,(a,b)) \qquad \text{since } \text{expose}_{\mathsf{S}}(s) = (a,c).$$

$$= s \qquad \text{since } s \text{ is a } \begin{pmatrix}(a,b)\\(a,c)\end{pmatrix}\text{-steady state.}$$

$$\text{expose}_{\mathsf{S}'}(s) = \pi_2 \text{expose}_{\mathsf{S}}(s) = c.$$

This shows that s is also a $\binom{b}{c}$ steady state of S', giving us a function $(a,s) \mapsto s : (\text{tr}_A \text{Steady}_{\mathsf{S}}) \to \text{Steady}_{\mathsf{S}'}$. These two functions are plainly inverse. □

We can summarize Proposition 3.14 in the following commutative diagram:



(3.15)

The horizontal maps take the steady states of a system, while the vertical map on the left wires together the system with that wiring diagram, and the vertical map on the right applies that transformation of the matrix. In the next section, we will see how this square can be interepreted as a naturality condition in a *doubly indexed functor*.

One thing to notice here is that taking the partial trace (the right vertical arrow in the diagram) is itself given by multiplying by a certain matrix.

Proposition 3.16. Let M be a $(A \times C) \times (A \times B)$ matrix of sets. Let Tr^A be the $\big(C \times B\big) \times \big((A \times C) \times (A \times B)\big)$ matrix of sets with entries:

$$\mathsf{Tr}^A A_{(c,b)((a,c'),(a',b'))} := \begin{cases} 1 & \text{if } a = a',\ b = b', \text{ and } c = c'. \\ \emptyset & \text{otherwise.} \end{cases}$$

Then, considering M as a $\big((A \times C) \times (A \times B)\big) \times 1$ matrix of sets, taking its trace is given by multiplying by Tr^A:

$$\text{tr}_A M \cong \mathsf{Tr}^A M$$

Proof. Let's calculate that matrix product on the right.

$$(\mathsf{Tr}^A M)_{(c,b)} = \sum_{((a,c'),(a',b'))\in(A\times C)\times(A\times B)} \mathsf{Tr}^A_{(c,b)((a,c'),(a',b'))} \times M_{(a,c')(a',b')}$$

Now, since $\mathsf{Tr}^A_{(c,b)((a,c'),(a',b'))}$ is a one element set (if $a = a'$, $c = c'$, and $b = b'$) and is empty otherwise, the inner expression has the elements of $M_{(a,c')(a',b')}$ if and only if $a = a'$, $b = b'$, and $c = c'$ and is otherwise empty. So, we conclude that

$$\sum_{((a,c'),(a',b'))\in(A\times C)\times(A\times B)} \mathsf{Tr}^A_{(c,b)((a,c'),(a',b'))} \times M_{(a,c')(a',b')} \cong M_{(a,c)(a,b)}.$$

As desired. □

3.3 Indexed categories and the Grothendieck construction

An *indexed category* is a category which varies functorially over the objects of another category.

Definition 3.17. An *indexed category* $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ is a contravariant (pseudo-) functor[a]. We call the category $\mathcal{C}$ the *base* of the indexed category $\mathcal{A}$. Explicitly, an indexed category $\mathcal{A}$ has:

- A base category $\mathcal{C}$.
- For every object $C \in \mathcal{C}$ of the base, a category $\mathcal{A}(C)$.
- For every map $f : C \to C'$ in the base, a *pullback* functor $f^* : \mathcal{A}(C') \to \mathcal{A}(C)$, which we think of as "reindexing" the objects of $\mathcal{A}(C')$ so that they live over $\mathcal{A}(C)$.
- (Psuedo-functoriality) For any two functions $f : C \to C'$ and $g : C' \to C''$, we need a natural isomorphism $f^* \circ g^* \overset{\gamma_{g,f}}{=\!=} (g \circ f)^*$. And for any object C, we need a natural isomorphism : $\mathrm{id}_{\mathcal{A}(C)} \overset{\iota_C}{=\!=} (\mathrm{id}_C)^*$. These are required to satisfy a few *coherence conditions*:

$$\gamma_{h,gf} \circ (\gamma_{g,f} h^*) = \gamma_{hg,f} \circ (f^* \gamma_{h,g}) \tag{3.18}$$

$$\gamma_{\mathrm{id}_C,f} \circ (f^* \iota_C) = \gamma_{f,\mathrm{id}_{C'}} \circ (\iota_{C'} f^*). \tag{3.19}$$

Often, γ and ι will both be identities, and these equalities will follow trivially.

[a] A pseudo-functor is like a functor, but it only satisfies the functoriality conditions up to isomorphism, and these isomorphisms must satisfy some laws that make them "cohere". The indexed categories we will see in Chapters 1 to 3 will all be actual functors, however.

Remark 3.20. The coherence conditions are almost always trivial — if not strictly identities, then they are probably just a shuffling of parentheses. For this reason, and because keeping track of them is a headache, we will omit discussion of the coherences. We

will write a coherence with a big equals, like this $f^* \circ g^* == (g \circ f)^*$ and leave the name implicit. The coherence conditions ensure that working with these isomorphisms as though they really were equalities works well.

Indexed categories are quite common throughout mathematics. We will construct a particular example for our own purposes in Section 3.4, and more throughout the book.

Example 3.21. Recall that a *dependent set* is a function $X : A \to \mathbf{Set}$ from a set into the category of sets. We have an indexed category of dependent sets

$$\mathbf{Set}^{(-)} : \mathbf{Set}^{\mathrm{op}} \to \mathbf{Cat}$$

which is defined as follows:

- To each set A, we assign the category $\mathbf{Set}^A$ of sets indexed by A. The objects of $\mathbf{Set}^A$ are the sets $X : A \to \mathbf{Set}$ indexed by A, and a map $f : X \to Y$ is a family of maps $f_a : X_a \to Y_a$ indexed by the elements $a \in A$. Composition is given componentwise: $(g \circ f)_a = g_a \circ f_a$.
- To every function $f : A' \to A$, we get a reindexing functor

  $$f^* : \mathbf{Set}^A \to \mathbf{Set}^{A'}$$

  Given by precomposition: $X \mapsto X \circ f$. The indexed set $X \circ f : A' \to \mathbf{Set}$ is the set $X_{f(a')}$ on the index $a' \in A'$. The families of functions get reindexed the same way.
- Since our reindexing is just given by precomposition, it is clearly functorial on the nose (that is, not just up to isomorphism).

We will return to this example in much greater detail in **??**.

If we have an family of sets $A : I \to \mathbf{Set}$ indexed by a set I, we can form the disjoint union $\sum_{i \in I} A_i$, together with the projection $\pi : \sum_{i \in I} A_i \to I$ sending each $a \in A_i$ to i. The Grothendieck construction is a generalization of this construction to indexed categories. Namely, we will take an indexed category $\mathcal{A} : \mathcal{C} \to \mathbf{Cat}$ and form a new category

$$\int^{C:\mathcal{C}} \mathcal{A}(C)$$

which we think of as a "union" off all the categories $\mathcal{A}(C)$. But this "union" will not be disjoint since there will be morphisms from objects in $\mathcal{A}(C)$ to objects in $\mathcal{A}(C')$. This is why we use the integral notation; we want to suggest that the Grothendieck construction is a sort of sum.[2]

[2]The Grothendieck construction is an example of a *lax colimit* in 2-category theory, another sense in which it is a 'sort of sum'.

Definition 3.22. Let $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ be an indexed category. The *Grothendieck construction* of $\mathcal{A}$

$$\int^{C:\mathcal{C}} \mathcal{A}(C)$$

is the category with:

- Objects pairs $\binom{A}{C}$ of objects $C \in \mathcal{C}$ and $A \in \mathcal{A}(C)$. We say that A "sits over" C.
- Maps $\binom{f_\flat}{f} : \binom{A}{C} \rightrightarrows \binom{A'}{C'}$ pairs of $f : C \to C'$ in $\mathcal{C}$ and $f_\flat : A \to f^*A'$ in $\mathcal{A}(C)$.
- Given $\binom{f_\flat}{f} : \binom{A}{C} \rightrightarrows \binom{A'}{C'}$ and $\binom{g_\flat}{g} : \binom{A'}{C'} \rightrightarrows \binom{A''}{C''}$, their composite is given by

$$\binom{g_\flat}{g} \circ \binom{f_\flat}{f} := \binom{f^*g_\flat \circ f_\flat}{g \circ f}$$

  Written with the signatures, this looks like

$$\begin{pmatrix} A \xrightarrow{f_\flat} f^*A' \xrightarrow{f^*g_\flat} f^*g^*A'' = (g \circ f)^*A'' \\ C \xrightarrow{f} C' \xrightarrow{g} C'' \end{pmatrix}$$

- The identity is given by $\binom{\mathrm{id}_A}{\mathrm{id}_C} : \binom{A}{C} \rightrightarrows \binom{A}{C}$

Exercise 3.23. Check that Definition 3.22 does indeed make $\int^{C:\mathcal{C}} \mathcal{A}(C)$ into a category. That is, check that composition as defined above is associative and unital. ◊

The confluence of notation with that of charts is no accident. Indeed, we will see in the next section that the category of charts can be described as the Grothendieck construction of a certain indexed category.

3.4 Sets with context, charts, and lenses

In this section, we will see how both the category **Chart** and **Lens** of charts and lenses in the category of sets can be described using the Grothendieck construction. To do this, we need some other categories named after their maps (rather than their objects): *category of sets and functions with context* C for some a given set C.

Definition 3.24. Let C be a set. The *category* $\boldsymbol{Ctx}_C$ *of sets and functions with context* C is the category defined by:

- Objects are sets.
- Maps $f : X \rightsquigarrow Y$ are functions $f : C \times X \to Y$.

- The composite $g \circ f$ of $f : X \rightsquigarrow Y$ and $g : Y \rightsquigarrow Z$ is the function

$$(c, x) \mapsto g(c, f(c, x)) : C \times X \to Z.$$

  Diagrammatically, this is the composite:

$$C \times X \xrightarrow{\Delta_C \times X} C \times C \times X \xrightarrow{C \times f} C \times Y \xrightarrow{g} Z.$$

- The identity $\mathrm{id} : X \rightsquigarrow X$ is the second projection $\pi_2 : C \times X \to X$.

Exercise 3.25. Check that $\mathbf{Ctx}_C$, as defined, really is a category. That is,

1. For $f : X \rightsquigarrow Y$, $g : Y \rightsquigarrow Z$, and $h : Z \rightsquigarrow W$, check that $h \circ (g \circ f) = (h \circ g) \circ f$.
2. For $f : X \rightsquigarrow Y$, check that $f \circ \mathrm{id}_X = f = \mathrm{id}_Y \circ f$.

◊

Together, we can arrange the categories of sets and functions with context into an indexed category.

Definition 3.26. The *indexed category of sets and functions with context*

$$\mathbf{Ctx}_{\_} : \mathbf{Set}^{\mathrm{op}} \to \mathbf{Cat}$$

is defined by:

- For a set C, we have the category $\mathbf{Ctx}_C$.
- For a function $r : C' \to C$, we get a functor

$$r^* : \mathbf{Ctx}_C \to \mathbf{Ctx}_{C'}$$

  given by sending each object to itself, but each morphism $f : C \times X \to Y$ in $\mathbf{Ctx}_C$ to the morphism $r^* f := f \circ (r \times X)$:

$$C' \times X \xrightarrow{r \times X} C \times X \xrightarrow{f} Y.$$

We note that this is evidently functorial.

Now we can see why the category of charts and the category of lenses are so eerily similar: one will be the Grothendieck construction of $\mathbf{Ctx}_{\_}$, and the other will be the Grothendieck construction of its dual $\mathbf{Ctx}_{\_}^{\mathrm{op}}$.

Proposition 3.27. The category **Chart** of charts in the category of sets (Definition 2.34) is the Grothendieck construction of $\mathbf{Ctx}_{\_} : \mathbf{Set}^{\mathrm{op}} \to \mathbf{Cat}$:

$$\mathbf{Chart} = \int^{C \in \mathbf{Set}} \mathbf{Ctx}_C.$$

Proof. We will expand the definition of the Grothendieck construction, and see that it gives us precisely Definition 2.34.

First, the objects of $\int^{C \in \mathbf{Set}} \mathbf{Ctx}_C$ are pairs of sets $\begin{pmatrix} A^- \\ A^+ \end{pmatrix}$, since the objects of $\mathbf{Ctx}_{A^+}$ are just sets themselves. So far so good.

Next, a map $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} A^- \\ A^+ \end{pmatrix} \rightrightarrows \begin{pmatrix} B^- \\ B^+ \end{pmatrix}$ in $\int^{C \in \mathbf{Set}} \mathbf{Ctx}_C$ is a pair of maps $f : A^+ \to B^+$ and $f_\flat : A^- \rightsquigarrow f^* B^-$ in $\mathbf{Ctx}_{A^+}$. But $f^* B^- = B^-$ by the definition of the reindexing functor f^*, so by the definition of map with context A^+, we see that $f_\flat : A^+ \times A^- \to B^-$. In other words, the maps in $\int^{C \in \mathbf{Set}} \mathbf{Ctx}_C$ are precisely the charts. We note that the identity map in the Grothendieck construction is the identity chart.

Finally, we should check that composition of charts is given by composition in the Grothendieck construction. Suppose that $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} A^- \\ A^+ \end{pmatrix} \rightrightarrows \begin{pmatrix} B^- \\ B^+ \end{pmatrix}$ and $\begin{pmatrix} g_\flat \\ g \end{pmatrix} : \begin{pmatrix} B^- \\ B^+ \end{pmatrix} \rightrightarrows \begin{pmatrix} C^- \\ C^+ \end{pmatrix}$ are charts. Then their composite in $\int^{C \in \mathbf{Set}} \mathbf{Ctx}_C$ is given by

$$\begin{pmatrix} f^* g_\flat \circ f_\flat \\ g \circ f \end{pmatrix}.$$

Well, the bottom is all good, what about the top? Expanding the definition, we see that $f^* g_\flat = g_\flat \circ (f \times B^-)$, so that $f^* g_\flat \circ f_\flat$ is the map given by

$$(a^+, a^-) \mapsto g_\flat(f(a^+), f_\flat(a^+, a^-)),$$

which is precisely how we defined composition of charts! □

The only difference between the category of charts and the category of lenses is that for lenses, the maps on top go backwards.

Proposition 3.28. The category **Lens** of lenses in the category of sets is the Grothendieck construction of the indexed category of *opposites* of the categories of sets and functions with context:

$$\mathbf{Lens} = \int^{C \in \mathbf{Set}} \mathbf{Ctx}_C^{\mathrm{op}}.$$

Proof. As with Proposition 3.27, we will simply expand the definition of the right hand side, and see that it is precisely the category of lenses.

The objects of $\int^{C \in \mathbf{Set}} \mathbf{Ctx}_C^{\mathrm{op}}$ are pairs $\begin{pmatrix} A^- \\ A^+ \end{pmatrix}$ of sets. All good so far.

A map in $\int^{C \in \mathbf{Set}} \mathbf{Ctx}_C^{\mathrm{op}}$ is a pair $\begin{pmatrix} f^\sharp \\ f \end{pmatrix}$ with $f : A^+ \to B^+$ and $f^\sharp : A^- \rightsquigarrow f^* B^-$ in $\mathbf{Ctx}_{A^+}^{\mathrm{op}}$. Now, $f^* B^- = B^-$ so $f^\sharp$ has signature $A^- \rightsquigarrow B^-$ in $\mathbf{Ctx}_{A^+}^{\mathrm{op}}$, which means $f^\sharp$ has signature $B^- \rightsquigarrow A^-$ in $\mathbf{Ctx}_{A^+}$, which means that $f^\sharp$ is a really a function $A^+ \times B^- \to A^-$. In other words, a map in $\int^{C \in \mathbf{Set}} \mathbf{Ctx}_C^{\mathrm{op}}$ is precisely a lens. We note that the identity map is the identity lens.

Finally, we need to check that composition in $\int^{C \in \mathbf{Set}} \mathbf{Ctx}_C^{\mathrm{op}}$ is lens composition. Suppose that $\begin{pmatrix} f^\sharp \\ f \end{pmatrix} : \begin{pmatrix} A^- \\ A^+ \end{pmatrix} \leftrightarrows \begin{pmatrix} B^- \\ B^+ \end{pmatrix}$ and $\begin{pmatrix} g^\sharp \\ g \end{pmatrix} : \begin{pmatrix} B^- \\ B^+ \end{pmatrix} \leftarrow \begin{pmatrix} C^- \\ C^+ \end{pmatrix}$ are lenses. In $\int^{C \in \mathbf{Set}} \mathbf{Ctx}_C^{\mathrm{op}}$,

their composite is

$$\begin{pmatrix} f^* g^\sharp \circ f^\sharp \\ g \circ f \end{pmatrix}.$$

The bottom is all good, we just need to check that the top — which, remember, lives in $\mathbf{Ctx}^{\mathrm{op}}_{A^+}$ — is correct. Since the composite up top is in the opposite, we are really calculating $f^\sharp \circ f^* g^\sharp$ in $\mathbf{Ctx}_{A^+}$. By definition, this is

$$(a^+, c^-) \mapsto f^\sharp(a^+, g^\sharp(f(a^+), c^-))$$

which is precisely their composite as lenses! □

Exercise 3.29. Make sure you *really* understand Propositions 3.27 and 3.28. ◊

Inspired by these two propositions, we will make the following general definitions.

Definition 3.30. Let $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ be an indexed category.

1. The category of $\mathcal{A}$-charts is the Grothendieck construction of $\mathcal{A}$:

$$\mathbf{Chart}_{\mathcal{A}} = \int^{C \in \mathcal{C}} \mathcal{A}(C).$$

2. The category of $\mathcal{A}$-lenses is the Grothendieck construction of $\mathcal{A}^{\mathrm{op}}$:

$$\mathbf{Lens}_{\mathcal{A}} = \int^{C \in \mathcal{C}} \mathcal{A}(C)^{\mathrm{op}}.$$

Sections of indexed categories. Any function $f : A \to B$ gives rise to a chart $\begin{pmatrix} f \circ \pi_2 \\ f \end{pmatrix} : \begin{pmatrix} A \\ A \end{pmatrix} \rightrightarrows \begin{pmatrix} B \\ B \end{pmatrix}$ by simply ignoring the context A. This gives us a functor $\mathbf{Set} \to \mathbf{Chart}$ which we call a *section* of the indexed category $\mathbf{Ctx}_-$.

Definition 3.31. Let $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ be an indexed category. A *section* T of $\mathcal{A}$ is a functor $\begin{pmatrix} T- \\ - \end{pmatrix} : \mathcal{C} \to \int^{C \in \mathcal{C}} \mathcal{A}(C)$ so that the bottom component of the image $\begin{pmatrix} Tf \\ f \end{pmatrix}$ of any map f in $\mathcal{C}$ is f itself.

Proposition 3.32. The functor $\begin{pmatrix} - \circ \pi_2 \\ - \end{pmatrix} : \mathbf{Set} \to \mathbf{Chart}$ is a section of the indexed category $\mathbf{Ctx}_-$.

Proof. By definition, the bottom component of $\begin{pmatrix} f \circ \pi_2 \\ f \end{pmatrix}$ is f, so we are really just checking that this is a functor. We note that it sends identities to identities.

Let $f : A \to B$ and $g : B \to C$ be functions. We need to show that

$$\begin{pmatrix} g \circ \pi_2 \\ g \end{pmatrix} \circ \begin{pmatrix} f \circ \pi_2 \\ f \end{pmatrix} = \begin{pmatrix} (g \circ f) \circ \pi_2 \\ g \circ f \end{pmatrix}.$$

The chart composite of $g \circ \pi_2$ with $f \circ \pi_2$ sends (a, a') to $g(\pi_2(f(a), f(\pi_2(a, a'))))$, which we can quickly see equals $g(f(a'))$ which is $(g \circ f)(\pi_2(a, a'))$. □

Remark 3.33. Sections of indexed categories are very useful in the theory of dynamical systems because they give us a notion of "changes possible in a given state". The section of **??** is a way of telling us that in a deterministic system, a system can transition to any state from any state. We'll see more of this point of view when we formally define dynamical system doctrines in Chapter 2.

Pure and cartesian maps. A map in a Grothendieck construction is a pair $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} A \\ C \end{pmatrix} \rightrightarrows \begin{pmatrix} A' \\ C' \end{pmatrix}$ of maps $f : C \to C'$ and $f_\flat : A \to f^* A'$. It is not too hard to see that a map is an isomorphism in a Grothendieck construction if and only if both its constituent maps are isomorphisms in their respective categories.

Proposition 3.34. Let $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ be an indexed category and let $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} A \\ C \end{pmatrix} \rightrightarrows \begin{pmatrix} A' \\ C' \end{pmatrix}$ be a map in its Grothendieck construction. Then $\begin{pmatrix} f_\flat \\ f \end{pmatrix}$ is an isomorphism if and only if f is an isomorphism in $\mathcal{C}$ and $f_\flat$ is an isomorphism in $\mathcal{A}(C)$.

Proof. First, let's show that if both f and $f_\flat$ are isomorphisms, then $\begin{pmatrix} f_\flat \\ f \end{pmatrix}$ is an isomorphism. We then have $f^{-1} : C' \to C$ and $f_\flat^{-1} : f^* A' \to A$. From $f_\flat^{-1}$, we can form $(f^{-1})^*(f_\flat^{-1}) : (f^{-1})^* f^* A' \to (f^{-1})^* A$, which we can pre-compose with some of the coherences to have the signature $A' \to (f^{-1})^* A$:

$$A' = (f \circ f^{-1})^* A' = (f^{-1})^* f^* A' \xrightarrow{(f^{-1})^*(f_\flat^{-1})} (f^{-1})^* A.$$

Now, consider the map $\begin{pmatrix} (f^{-1})^* f_\flat^{-1} \\ f^{-1} \end{pmatrix} : \begin{pmatrix} A' \\ C' \end{pmatrix} \rightrightarrows \begin{pmatrix} A \\ C \end{pmatrix}$. We'll show that this is an inverse to $\begin{pmatrix} f_\flat \\ f \end{pmatrix}$. Certainly, the bottom components will work out; we just need to worry about the top. That is, we need to show that $f^*((f^{-1})^* f_\flat^{-1}) \circ f_\flat = \mathrm{id}$ and $(f^{-1})^*(f_\flat) \circ (f^{-1})^*(f_\flat^{-1}) = \mathrm{id}$. Both of these follow quickly by functoriality.

On the other hand, suppose that $\begin{pmatrix} f_\flat \\ f \end{pmatrix}$ is an isomorphism with inverse $\begin{pmatrix} g_\flat \\ g \end{pmatrix}$. Then $gf = \mathrm{id}$ and $fg = \mathrm{id}$, so f is an isomorphism. We can focus on $f_\flat$. We know that $f^* g_\flat \circ f_\flat = \mathrm{id}$ and $g^* f_\flat \circ g_\flat = \mathrm{id}$. Applying f^* to the second equation, we find that $f_\flat \circ f^* g_\flat = \mathrm{id}$, so that $f_\flat$ is an isomorphism with inverse $f^* g_\flat$. □

Remark 3.35. Proposition 3.34 gives a general solution to Exercise 2.42, since the category of charts is a Grothendieck construction.

This proposition suggests two interesting classes of maps in a Grothendieck construction: the maps $\begin{pmatrix} f_\flat \\ f \end{pmatrix}$ for which f is an isomorphism, and those for which $f_\flat$ is an isomorphism.

Definition 3.36. Let $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ be an indexed category and let $\begin{pmatrix} f_\flat \\ f \end{pmatrix}$ be a map in its Grothendieck construction. We say that $\begin{pmatrix} f_\flat \\ f \end{pmatrix}$ is

- *pure* if f is an isomorphism, and
- *cartesian* if $f_\flat$ is an isomorphism.

The pure maps correspond essentially to the maps in the categories $\mathcal{A}(C)$ at a given index C, while the cartesian maps correspond essentially to the maps in $\mathcal{C}$.

Remark 3.37. The name "pure" is non-standard. The usual name is "vertical". But we are about to talk about "vertical" maps in a technical sense when we come to double categories, so we've renamed the concept here to avoid confusion later.

Example 3.38. We have often seen systems that expose their entire state, like Time of Example 2.28. Considered as lenses, these are *pure* in the sense that their expose function is an isomorphism.

Exercise 3.39. Let $\begin{pmatrix} f_\flat \\ f \end{pmatrix}$ and $\begin{pmatrix} g_\flat \\ g \end{pmatrix}$ be composable maps in a Grothendieck construction,

1. Suppose that $\begin{pmatrix} g_\flat \\ g \end{pmatrix}$ is cartesian. Show that $\begin{pmatrix} f_\flat \\ f \end{pmatrix}$ is cartesian if and only if their composite is cartesian. Is the same true for pure maps?
2. Suppose that $\begin{pmatrix} f_\flat \\ f \end{pmatrix}$ is pure. Show that $\begin{pmatrix} g_\flat \\ g \end{pmatrix}$ is pure if and only if their composite is pure. Is the same true for cartesian maps?

◊

3.5 Double categories of matrices and profunctors

3.5.1 The double category of arenas, charts, and lenses

We've seen a bit of what we can do with the double category of arenas, but it does remain for us to show that Definition 2.53 really gives a double category. We could do this by hand, but we'll do it diagrammatically by generalizing the double category of arenas to any indexed category.

Definition 3.40. Let $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ be an indexed category. The *Grothendieck double construction*

$$\iint^{C \in \mathcal{C}} \mathcal{A}(C)$$

is the double category defined by:

- Its objects are the pairs $\binom{A}{C}$ of an object $C \in \mathcal{C}$ and an object $A \in \mathcal{A}(C)$.
- Its horizontal category is the Grothendieck construction $\int^{C \in \mathcal{C}} \mathcal{A}(C)$ of $\mathcal{A}$.
- Its vertical category is the Grothendieck construction $\int^{C \in \mathcal{C}} \mathcal{A}(C)^{\mathrm{op}}$ of the opposite of $\mathcal{A}$.
- There is a square of the following form:

$$\begin{array}{ccc} \binom{A_1}{C_1} & \overset{\binom{g_{1\flat}}{g_1}}{\Longrightarrow} & \binom{A_2}{C_2} \\ \binom{f_1^\sharp}{f_1}\downarrow\uparrow & & \downarrow\uparrow\binom{f_2^\sharp}{f_2} \\ \binom{A_3}{C_3} & \underset{\binom{g_{2\flat}}{g_2}}{\Longrightarrow} & \binom{A_4}{C_4} \end{array} \tag{3.41}$$

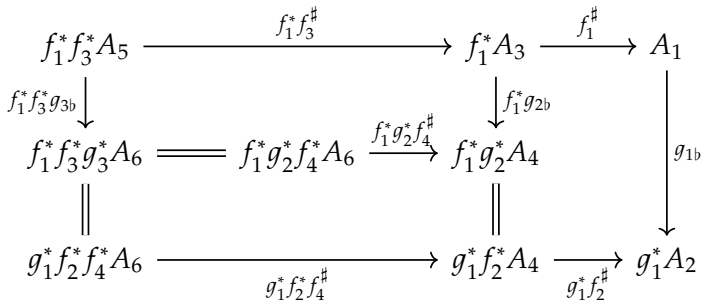
  if and only if the following diagrams commute:

$$\begin{array}{ccc} C_1 & \xrightarrow{g_1} & C_2 \\ {\scriptstyle f_1}\downarrow & & \downarrow{\scriptstyle f_2} \\ C_3 & \xrightarrow[g_2]{} & C_4 \end{array} \qquad \begin{array}{ccc} f_1^* A_3 & \xrightarrow{\qquad f_1^\sharp \qquad} & A_1 \\ {\scriptstyle f_1^* g_{2\flat}}\downarrow & & \downarrow{\scriptstyle g_{1\flat}} \\ f_1^* g_2^* A_4 & = & g_1^* f_2^* A_4 \xrightarrow[g_1^* f_2^\sharp]{} g_1^* A_2 \end{array} \tag{3.42}$$

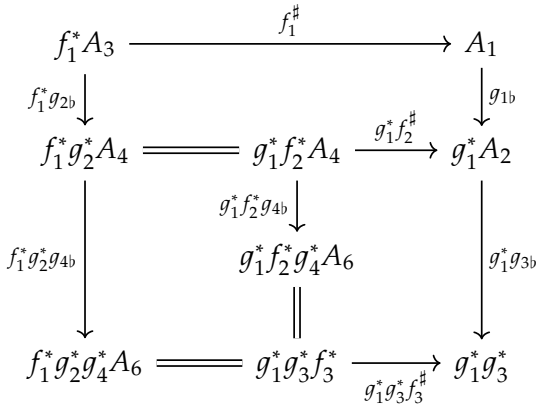
  We will call the squares in the Grothendieck double construction *commuting squares*, since they represent the proposition that the "lower" and "upper" squares appearing in their boundary commute.
- Composition is given as in the appropriate Grothendieck constructions.

It just remains to show that commuting squares compose.

- For vertical composition we appeal to the following diagram:



  The outer diagram is the "upper" square of the composite, while the "upper" squares of each factor appear in the top left and right respectively.
- For horizontal composition we appeal to the following diagram:



We can now check that this does indeed abstract the double category of arenas.

Proposition 3.43. The double category of arenas in the deterministic doctrine is the Grothendieck double construction of the indexed category of sets and functions in context $\mathbf{Ctx}_{\_} : \mathbf{Set}^{\mathrm{op}} \to \mathbf{Cat}$:

$$\mathbf{Arena} = \iint^{C \in \mathbf{Set}} \mathbf{Ctx}_C.$$

Proof. By Propositions 3.27 and 3.28, the horizontal and vertical categories are the same. It remains to show that the diagrams of Eq. (3.41) mean the same things as Eq. (2.55).

Consider a square of the form

$$\begin{array}{ccc} \begin{pmatrix} A^- \\ A^+ \end{pmatrix} & \overset{\begin{pmatrix} f_\flat \\ f \end{pmatrix}}{\rightrightarrows} & \begin{pmatrix} B^- \\ B^+ \end{pmatrix} \\ \begin{pmatrix} j^\sharp \\ j \end{pmatrix} \downarrow\uparrow & & \downarrow\uparrow \begin{pmatrix} k^\sharp \\ k \end{pmatrix} \\ \begin{pmatrix} C^- \\ C^+ \end{pmatrix} & \underset{\begin{pmatrix} g^\sharp \\ g \end{pmatrix}}{\rightrightarrows} & \begin{pmatrix} D^- \\ D^+ \end{pmatrix} \end{array}$$

The first diagram and first equation say:

$$\begin{array}{ccc} A^+ & \xrightarrow{f} & B^+ \\ {\scriptstyle j}\downarrow & & \downarrow{\scriptstyle k} \\ C^+ & \xrightarrow[g]{} & D^+ \end{array} \qquad g(j(a^+)) = k(f(a^+)) \quad \text{for all } a^+ \in A^+,$$

which mean the same thing. The second diagram, which takes place in $\mathbf{Ctx}_{A^+}$, is more interesting. Here's that diagram with the names we're currently using:

$$\begin{array}{ccccc} j^*C^- & & \xrightarrow{\qquad\qquad j^\sharp \qquad\qquad} & & A^- \\ {\scriptstyle j^*g_\flat}\downarrow & & & & \downarrow{\scriptstyle f_\flat} \\ j^*g^*D^- & = & f^*k^*D^- & \xrightarrow[f^*k^\sharp]{} & f^*B^- \end{array}$$

Let's compute the two paths from the top left to the bottom right. First is $f_\flat \circ j^\sharp : j^*C^- \to f^*B^-$, which sends (a^+, c^-) to $f_\flat(a^+, j^\sharp(a^+, c^-))$. This is the right hand side of the second equation, so we're on the right track. The other path is $f^*k^\sharp \circ j^*g_\flat$. Recall that $j^*g_\flat$ sends (a^+, c^-) to $g_\flat(j(a^+), c^-)$, and similarly $f^*k^\sharp$ sends (a^+, d^-) to $k^\sharp(f(a^+), d^-)$. Putting them together, we send (a^+, c^-) to $k^\sharp(f(a^+), g_\flat(j(a^+), c^-))$. Therefore the commutation of this diagram means the same thing as the second equation in the definition of a square of arenas. □

Building off of this proposition, we can think of the Grothendieck double construction as giving us a double category of arenas out of *any* indexed category.

Definition 3.44. Let $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ be an indexed category. Then the category of $\mathcal{A}$-arenas is defined to be the Grothendieck double construction of $\mathcal{A}$:

$$\mathbf{Arena}_{\mathcal{A}} := \iint^{C \in \mathcal{C}} \mathcal{A}(C).$$

Note that the horizontal category of $\mathbf{Arena}_{\mathcal{A}}$ is the category $\mathbf{Chart}_{\mathcal{A}}$ of $\mathcal{A}$-charts (Item 1), and the vertical category of $\mathbf{Arena}_{\mathcal{A}}$ is the category $\mathbf{Lens}_{\mathcal{A}}$ of $\mathcal{A}$-lenses (Item 2).

While we're busy generalizing, let's go a bit deeper. Note that in order to make the square in Eq. (2.59), we had to use the section $\binom{-\circ\pi_2}{-}$ we constructed in Proposition 3.32 to turn the function $\phi : \mathsf{State}_\mathsf{T} \to \mathsf{State}_\mathsf{S}$ into a chart $\binom{\phi\circ\pi_2}{\phi} : \binom{\mathsf{State}_\mathsf{T}}{\mathsf{State}_\mathsf{T}} \rightrightarrows \binom{\mathsf{State}_\mathsf{S}}{\mathsf{State}_\mathsf{S}}$. This observation lets us generalize the above definition of categories of systems (from Definition 2.43) to *any indexed category with a section*!

Definition 3.45. Let $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ be an indexed category, and let $\binom{T-}{-} : \mathcal{C} \to \mathbf{Chart}_{\mathcal{A}}$ be a section. For a $\mathcal{A}$-arena $\binom{A}{C}$, the category $\mathbf{Sys}\binom{A}{C}$ of $(\mathcal{A}, T)$-systems with interface $\binom{A}{C}$ is defined by:

- Its objects are $\mathcal{A}$-lenses $\binom{\mathrm{update}_\mathsf{S}}{\mathrm{expose}_\mathsf{S}} : \binom{T\mathsf{State}_\mathsf{S}}{\mathsf{State}_\mathsf{S}} \leftrightarrows \binom{A}{C}$, which we can call *systems* in this general setting.
- Its maps are squares

$$\begin{array}{ccc} \binom{\mathsf{State}_\mathsf{T}}{\mathsf{State}_\mathsf{T}} & \xRightarrow{\binom{T\phi}{\phi}} & \binom{\mathsf{State}_\mathsf{S}}{\mathsf{State}_\mathsf{S}} \\ \binom{\mathrm{update}_\mathsf{T}}{\mathrm{expose}_\mathsf{T}} \downarrow\uparrow & & \downarrow\uparrow \binom{\mathrm{update}_\mathsf{S}}{\mathrm{expose}_\mathsf{S}} \\ \binom{A}{C} & = & \binom{A}{C} \end{array}$$

- Composition is given by horizontal composition in the double category $\mathbf{Arena}_{\mathcal{A}}$ of $\mathcal{A}$-arenas.

3.5.2 The double category of sets, functions, and matrices

Now we turn to our second double category of interest, the double category of sets, functions, and matrices of sets.

Definition 3.46. The double category **Matrix** of sets, functions, and matrices of sets is defined by:

- Its objects are sets.
- Its horizontal category is the category of sets and functions.
- Its vertical category is the category of sets and matrices of sets, where composition is given by matrix multiplication. We write $M : A \to B$ to say that M is a $B \times A$ matrix.[a]
- For functions $f : A \to B$ and $g : C \to D$ and matrices $M : A \to C$ and $N : B \to D$,

a square

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle M}\downarrow & \alpha & \downarrow{\scriptstyle N} \\ C & \xrightarrow[g]{} & D \end{array}$$

is a family of functions $\alpha_{ba} : M_{ba} \to N_{g(b)f(a)}$ for all $a \in A$ and $b \in B$.

- Horizontal composition of squares is given by composition of the families:

$$(\alpha|\beta)_{ba} = \beta_{g(b)f(a)} \circ \alpha_{ba}.$$

- Vertical composition of squares is given by

$$\left(\frac{\alpha}{\beta}\right)_{ac} : \sum_{b_1 \in B_1} M^2_{cb} \times M^1_{ba} \to \sum_{b_2 \in B_2} N^2_{h(c)b_2} \times N^1 b_2 f(a)$$
$$(b_1, m_2, m_1) \mapsto (g(b_1), \beta(m_2), \alpha(m_1)).$$

[a] Observant readers will notice that multiplication of matrices of sets isn't strictly associative or unital. It only satisfies those properties up to isomorphism. Technically, we are defining a form of *weak* double category, where one form of composition is only defined up to isomorphism. But since these isomorphisms are a trivial form of book-keeping, we'll sweep this inconvenient fact under the rug.

Exercise 3.47. We can see that horizontal composition of squares is associative and unital since it is basically just function composition. Show that **Matrix** is a double category by checking that

1. Vertical composition of squares is associative and unital (up to isomorphism).
2. The interchange law holds.

◊

3.5.3 The double category of categories, profunctors, and functors

Now we come to the primordial double category: the double category of categories, *profunctors*, and functors. This is an important double category because it is in some sense the setting in which all category theory takes place. Before we describe this double category, let's define the notion of profunctor and their category.

Definition 3.48. A *profunctor* $P : \mathcal{A} \nrightarrow \mathcal{B}$ is a functor $P : \mathcal{A}^{\mathrm{op}} \times \mathcal{B} \to \mathbf{Set}$. Given objects $A \in \mathcal{A}$ and $B \in \mathcal{B}$, we write an element $p \in P(A, B)$ as $p : A \nrightarrow B$.

In terms of this, the functoriality of P can be seen as letting us compose $p : A \nrightarrow B$ on the left and right by $f : A' \to A$ and $g : B \to B'$ to get $fpg : A' \nrightarrow B'$. In other words, we can interpret a diagram of this form

$$A' \xrightarrow{f} A \overset{p}{\nrightarrow} B \xrightarrow{g} B'$$

as an element of $P(A', B')$.

If we call maps $f : A' \to A$ in a category $\mathcal{A}$ *homomorphisms* because they go between objects of the same form, we could call elements $p : A \overset{B}{\nrightarrow}$ — that is, $p \in P(A, B)$ — as *heteromorphisms*, maps going between objects of different forms.

We can't necessarily compose these heteromorphisms, which we can see right away from their signature: for $p : A \nrightarrow B$, there is always an object of $\mathcal{A}$ on the left and an object of $\mathcal{B}$ on the right, so we'll never be able to line two of them up. However, if we have another profunctor $Q : \mathcal{B} \nrightarrow \mathcal{C}$ — another notion of heteromorphism — then we can "compose" heteromorphisms $A \overset{p}{\nrightarrow} B$ in P with $B \overset{q}{\nrightarrow} C$ in Q to get a heteromorphism $A \overset{p}{\nrightarrow} B \overset{q}{\nrightarrow} C$ in a new profunctor $P \odot Q : \mathcal{A} \nrightarrow \mathcal{C}$.

Definition 3.49. The composite $P \odot Q$ of a profunctor $P : \mathcal{A} \nrightarrow \mathcal{B}$ with a profunctor $Q : \mathcal{B} \nrightarrow \mathcal{C}$ is defined to be the following quotient:

$$(P \odot Q)(A, C) := \frac{\sum_{B \in \mathcal{B}} P(A, B) \times Q(B, C)}{(pf, q) \sim (p, fq)} \tag{3.50}$$

We write an element $[(p, q)] \in (P \odot Q)(A, C)$ as $A \overset{p}{\nrightarrow} B \overset{q}{\nrightarrow} C$, so that the relation we quotient by says that

$$A \overset{p}{\nrightarrow} B \overset{f}{\to} B' \overset{q}{\nrightarrow} C$$

has a unique interpretation as an element of $P \odot Q$.

The identity profunctor $\mathcal{A} : \mathcal{A} \nrightarrow \mathcal{A}$ is the hom-functor sending A and A' to the set $\mathcal{A}(A, A')$ of maps $A \to A'$.

We can see that composition of profunctors is associative (up to isomorphism) because the objects of $P \odot (Q \odot R)$ and $(P \odot Q) \odot R$ can both be written as

$$A \overset{p}{\nrightarrow} B \overset{q}{\nrightarrow} C \overset{r}{\nrightarrow} D.$$

The reason the hom profunctor $\mathcal{A} : \mathcal{A} \nrightarrow \mathcal{A}$ is the identity profunctor is because the elements of $\mathcal{A} \odot P$ would be written as

$$A' \overset{f}{\to} A \overset{p}{\nrightarrow} B$$

but by the functoriality of P, this is already an element of $P(A', B)$, which is to say more precisely that every equivalence class $[(f, p)] \in (\mathcal{A} \odot P)(A', B)$ is equally presented as $[(\mathrm{id}_{A'}, fp)]$.

Exercise 3.51. Let $P : \mathcal{A} \nrightarrow \mathcal{B}$ be a profunctor.

1. Show that there is a natural transformation $\mathcal{A} \odot P \to P$ given by the naturality of P on the left.
2. Show that there is a natural transformation $P \odot \mathcal{B} \to P$ given by the naturality of

P on the right.

3. Show that both of these natural transformations are isomorphisms.

◊

Remark 3.52. We omit full proofs of associativity and unitality for profunctor composition because they are best done with the *coend calculus*, and this would take us quite far afield.

Example 3.53. A profunctor $1 \nrightarrow \mathcal{A}$ is the same thing as a functor $\mathcal{A} \to \mathbf{Set}$, and a profunctor $\mathcal{A} \nrightarrow 1$ is the same thing as a functor $\mathcal{A}^{\mathrm{op}} \to \mathbf{Set}$. Profunctors are therefore intimately related with presheaves.

Example 3.54. The notion of profunctor gives us a nice way to understand the relationship between a behavior $\phi : \mathsf{T} \to \mathsf{S}$ and its chart $\binom{f_\flat}{f} : \binom{I}{O} \rightrightarrows \binom{I'}{O'}$.

As we saw in Definition 2.43, for every arena $\binom{I}{O}$ we have a category $\mathbf{Sys}\binom{I}{O}$ of systems with interface $\binom{I}{O}$. Given a chart $\binom{f_\flat}{f} : \binom{I}{O} \rightrightarrows \binom{I'}{O'}$, we get a profunctor

$$\mathbf{Sys}\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \mathbf{Sys}\begin{pmatrix} I \\ O \end{pmatrix} \nrightarrow \mathbf{Sys}\begin{pmatrix} I' \\ O' \end{pmatrix}$$

Which is defined like this:

$$\mathbf{Sys}\begin{pmatrix} f_\flat \\ f \end{pmatrix}(\mathsf{T},\mathsf{S}) = \left\{ \phi : \mathsf{State}_\mathsf{T} \to \mathsf{State}_\mathsf{S} \;\middle|\; \phi \text{ is a behavior with chart } \begin{pmatrix} f_\flat \\ f \end{pmatrix} \right\}$$

$$= \left\{ \begin{array}{ccc} & \overset{\binom{\phi\circ\pi_2}{\phi}}{\dashrightarrow} & \\ \binom{\mathsf{State}_\mathsf{T}}{\mathsf{State}_\mathsf{T}} & & \binom{\mathsf{State}_\mathsf{S}}{\mathsf{State}_\mathsf{S}} \\ \binom{\mathsf{update}_\mathsf{T}}{\mathsf{expose}_\mathsf{T}} \downarrow\uparrow & & \downarrow\uparrow \binom{\mathsf{update}_\mathsf{S}}{\mathsf{expose}_\mathsf{S}} \\ \binom{\mathsf{In}_\mathsf{T}}{\mathsf{Out}_\mathsf{T}} & \underset{\binom{f^\sharp}{f}}{\Longrightarrow} & \binom{\mathsf{In}_\mathsf{S}}{\mathsf{Out}_\mathsf{S}} \end{array} \right\}$$

The action of the profunctor $\mathbf{Sys}\binom{f_\flat}{f}$ on behaviors in the categories $\mathbf{Sys}\binom{I}{O}$ and $\mathbf{Sys}\binom{I'}{O'}$ is given by composition on the left and right.

Now, we are ready to put functors and profunctors together into a double category.

Definition 3.55. The double category **Cat** of categories, profunctors, and functors has

- Objects the categories.
- Horizontal category the category of categories and profunctors.
- Vertical category the category of categories and functors between them.
- A square

$$\begin{array}{ccc} \mathcal{A} & \xrightarrow{\ P\ } & \mathcal{B} \\ {\scriptstyle F}\downarrow & \alpha & \downarrow{\scriptstyle G} \\ \mathcal{C} & \xrightarrow[\ Q\ ]{} & \mathcal{D} \end{array}$$

  Is a natural transformation $\alpha : P \to Q(F, G)$, where $Q(F, G)$ is the profunctor $\mathcal{A}^{\mathrm{op}} \times \mathcal{B} \xrightarrow{F^{\mathrm{op}} \times G} \mathcal{C} \times \mathcal{D} \xrightarrow{Q} \mathbf{Set}$. For $p : A \nrightarrow B$, we have $\alpha(p) : FA \nrightarrow GB$, and naturality says that $\alpha(fpg) = (Ff)\alpha(p)(Gg)$.
- Vertical composition of squares is given by composing the natural transformations.
- Given squares $\alpha : P_1 \to Q_1(F_1, F_2)$ and $\beta : P_2 \to Q_2(F_2, F_3)$, we define their horizontal composite $\alpha \mid \beta : P_1 \cdot P_2 \to (Q_1 \cdot Q_2)(F_1, F_3)$ by

$$(\alpha \mid \beta)(A_1 \overset{p_1}{\nrightarrow} A_2 \overset{p_2}{\nrightarrow} A_3) = F_1A_1 \overset{\alpha(p_1)}{\nrightarrow} F_2A_2 \overset{\beta(p_2)}{\nrightarrow} F_3A_3$$

  and checking that this descends correctly to the quotient.

Remark 3.56. We are using "**Cat**" to refer to the category of categories and functors and to the double category of categories, profunctors, and functors. The one we mean will be clear from context, and the category of categories and functors is the vertical category of the double category of categories.

Example 3.57. With a little work in the double category of arenas, we can give a very useful example of a square in the double category of profunctors. Consider this square in the double category of arenas:

$$\alpha = \begin{array}{ccc} \begin{pmatrix} I_1 \\ O_1 \end{pmatrix} & \overset{\binom{f^\sharp}{f}}{\Longrightarrow} & \begin{pmatrix} I_2 \\ O_2 \end{pmatrix} \\ \begin{pmatrix} j^\sharp \\ j \end{pmatrix}\downarrow\uparrow & & \downarrow\uparrow\begin{pmatrix} k^\sharp \\ k \end{pmatrix} \\ \begin{pmatrix} I_3 \\ O_3 \end{pmatrix} & \underset{\binom{g^\sharp}{g}}{\Longrightarrow} & \begin{pmatrix} I_4 \\ O_4 \end{pmatrix} \end{array}$$

As we saw in Proposition 2.65, we get functors $\mathbf{Sys}\left(\begin{smallmatrix} j^\sharp \\ j \end{smallmatrix}\right) : \mathbf{Sys}\left(\begin{smallmatrix} I_1 \\ O_1 \end{smallmatrix}\right) \to \mathbf{Sys}\left(\begin{smallmatrix} I_3 \\ O_3 \end{smallmatrix}\right)$ and $\mathbf{Sys}\left(\begin{smallmatrix} k^\sharp \\ k \end{smallmatrix}\right) : \mathbf{Sys}\left(\begin{smallmatrix} I_2 \\ O_2 \end{smallmatrix}\right) \to \mathbf{Sys}\left(\begin{smallmatrix} I_4 \\ O_4 \end{smallmatrix}\right)$ given by composing with these lenses. We also saw in Example 3.54 that we get profunctors $\mathbf{Sys}\left(\begin{smallmatrix} f_\flat \\ f \end{smallmatrix}\right) : \mathbf{Sys}\left(\begin{smallmatrix} I_1 \\ O_1 \end{smallmatrix}\right) \nrightarrow \mathbf{Sys}\left(\begin{smallmatrix} I_2 \\ O_2 \end{smallmatrix}\right)$ and $\mathbf{Sys}\left(\begin{smallmatrix} g_\flat \\ g \end{smallmatrix}\right) : \mathbf{Sys}\left(\begin{smallmatrix} I_3 \\ O_3 \end{smallmatrix}\right) \nrightarrow \mathbf{Sys}\left(\begin{smallmatrix} I_4 \\ O_4 \end{smallmatrix}\right)$ from these charts. In this example, we'll observe that we can get a square of profunctors

$$\mathbf{Sys}(\alpha) : \mathbf{Sys}\begin{pmatrix} f_\flat \\ f \end{pmatrix} \to \mathbf{Sys}\begin{pmatrix} g_\flat \\ g \end{pmatrix}\left(\mathbf{Sys}\begin{pmatrix} j^\sharp \\ j \end{pmatrix}, \mathbf{Sys}\begin{pmatrix} k^\sharp \\ k \end{pmatrix}\right)$$

from the square α in the double category of arenas.

To define the natural transformation $\mathbf{Sys}(\alpha)$, we need to say what it does to an element ϕ of $\mathbf{Sys}\left(\begin{smallmatrix} f_\flat \\ f \end{smallmatrix}\right)(\mathsf{T}, \mathsf{S})$. Recall that the elements of this profunctor are behaviors with chart $\left(\begin{smallmatrix} f_\flat \\ f \end{smallmatrix}\right)$, so really ϕ is a square

$$\phi = \begin{array}{ccc} \begin{pmatrix} \mathsf{State}_\mathsf{T} \\ \mathsf{State}_\mathsf{T} \end{pmatrix} & \overset{\left(\begin{smallmatrix} \phi \circ \pi_2 \\ \phi \end{smallmatrix}\right)}{\Longrightarrow} & \begin{pmatrix} \mathsf{State}_\mathsf{S} \\ \mathsf{State}_\mathsf{S} \end{pmatrix} \\ \left(\begin{smallmatrix} \mathsf{update}_\mathsf{T} \\ \mathsf{expose}_\mathsf{T} \end{smallmatrix}\right)\downarrow\uparrow & & \downarrow\uparrow\left(\begin{smallmatrix} \mathsf{update}_\mathsf{S} \\ \mathsf{expose}_\mathsf{S} \end{smallmatrix}\right) \\ \begin{pmatrix} \mathsf{In}_\mathsf{T} \\ \mathsf{Out}_\mathsf{T} \end{pmatrix} & \underset{\left(\begin{smallmatrix} f^\sharp \\ f \end{smallmatrix}\right)}{\Longrightarrow} & \begin{pmatrix} \mathsf{In}_\mathsf{S} \\ \mathsf{Out}_\mathsf{S} \end{pmatrix} \end{array}$$

in the double category of arenas. Therefore, we can define $\mathbf{Sys}(\alpha)(\phi)$ to be the vertical composite:

$$\begin{array}{ccc} \begin{pmatrix} \mathsf{State}_\mathsf{T} \\ \mathsf{State}_\mathsf{T} \end{pmatrix} & \overset{\left(\begin{smallmatrix} \phi \circ \pi_2 \\ \psi\phi \end{smallmatrix}\right)}{\Longrightarrow} & \begin{pmatrix} \mathsf{State}_\mathsf{S} \\ \mathsf{State}_\mathsf{S} \end{pmatrix} \\ \left(\begin{smallmatrix} \mathsf{update}_\mathsf{T} \\ \mathsf{expose}_\mathsf{T} \end{smallmatrix}\right)\downarrow\uparrow & & \downarrow\uparrow\left(\begin{smallmatrix} \mathsf{update}_\mathsf{S} \\ \mathsf{expose}_\mathsf{S} \end{smallmatrix}\right) \\ \begin{pmatrix} \mathsf{In}_\mathsf{T} \\ \mathsf{Out}_\mathsf{T} \end{pmatrix} & \underset{\left(\begin{smallmatrix} f^\sharp \\ f \end{smallmatrix}\right)}{\Longrightarrow} & \begin{pmatrix} \mathsf{In}_\mathsf{S} \\ \mathsf{Out}_\mathsf{S} \end{pmatrix} \\ \left(\begin{smallmatrix} j^\sharp \\ j \end{smallmatrix}\right)\downarrow\uparrow & & \downarrow\uparrow\left(\begin{smallmatrix} k^\sharp \\ k \end{smallmatrix}\right) \\ \begin{pmatrix} I \\ O \end{pmatrix} & \underset{\left(\begin{smallmatrix} g^\sharp \\ g \end{smallmatrix}\right)}{\Longrightarrow} & \begin{pmatrix} I' \\ O' \end{pmatrix} \end{array}$$

Or, a little more concisely in double category notation:

$$\mathbf{Sys}(\alpha)(\phi) = \frac{\phi}{\alpha}.$$

The naturality of this transformation follows from the double category laws. We leave the particulars as an exercise.

Exercise 3.58. Prove that the family of functions

$$\mathbf{Sys}(\alpha) : \mathbf{Sys}\begin{pmatrix} f_\flat \\ f \end{pmatrix} \to \mathbf{Sys}\begin{pmatrix} g_\flat \\ g \end{pmatrix}\left(\mathbf{Sys}\begin{pmatrix} j^\sharp \\ j \end{pmatrix}, \mathbf{Sys}\begin{pmatrix} k^\sharp \\ k \end{pmatrix}\right)$$

defined in Example 3.57 is a natural transformation. (Hint: use the double category notation, it will be much more concise.) ◊

3.6 Arranging categories along two kinds of composition: Doubly indexed categories

While we described a category of systems and behaviors in Proposition 2.37, we haven't been thinking of systems in quite this way. We have been organizing our systems a bit more particularly than just throwing them into one large category. We've made the following observations:

- Each system has an interface, and many different systems can have the same interface. From this observation, we defined the categories $\mathbf{Sys}\begin{pmatrix} I \\ O \end{pmatrix}$ of systems with the interface $\begin{pmatrix} I \\ O \end{pmatrix}$ in Definition 2.43.
- Every wiring diagram, or more generally lens, gives us an operation that changes the interface of a system by wiring things together. We formalized this observation into a functor $\begin{pmatrix} w^\sharp \\ w \end{pmatrix} : \mathbf{Sys}\begin{pmatrix} I \\ O \end{pmatrix} \to \mathbf{Sys}\begin{pmatrix} I' \\ O' \end{pmatrix}$ in Proposition 2.65.
- To describe the behavior of a system, first we have to chart out how it will look on its interface. We formalized this observation by giving a profunctor $\mathbf{Sys}\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \mathbf{Sys}\begin{pmatrix} I \\ O \end{pmatrix} \nrightarrow \mathbf{Sys}\begin{pmatrix} I' \\ O' \end{pmatrix}$ for each chart in Example 3.54.
- If we wire together a chart for one interface into a chart for the wired interface, then every behavior for that chart gives rise to a behavior for the wired together chart. We formalized this observation as a morphism of profunctors

$$\mathbf{Sys}(\alpha) : \mathbf{Sys}\begin{pmatrix} f_\flat \\ f \end{pmatrix} \to \mathbf{Sys}\begin{pmatrix} g_\flat \\ g \end{pmatrix}\left(\mathbf{Sys}\begin{pmatrix} j^\sharp \\ j \end{pmatrix}, \mathbf{Sys}\begin{pmatrix} k^\sharp \\ k \end{pmatrix}\right)$$

  in Example 3.57.

Now comes the time to organize all these observations. In this section, we will see that collectively, these observations are telling us that there is an *doubly indexed category*

of dynamical systems. We will also see that matrices of sets give rise to a doubly indexed category which we will call the doubly indexed category of vectors of sets.

Definition 3.59. A *doubly indexed category* $\mathcal{A} : \mathbb{D} \to \mathbf{Cat}$ consists of the following:[a]

- A double category $\mathbb{D}$ called the *indexing base*.
- For every object $D \in \mathbb{D}$, we have a category $\mathcal{A}(D)$.
- For every vertical arrow $j : D \to D'$, we have a functor $\mathcal{A}(j) : \mathcal{A}(D) \to \mathcal{A}(D')$.
- For every horizontal arrow $f : D \to D'$, we have a profunctor $\mathcal{A}(f) : \mathcal{A}(D) \nrightarrow \mathcal{A}(D')$.
- For every square

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle j}\downarrow & \alpha & \downarrow{\scriptstyle k} \\ C & \xrightarrow[g]{} & D \end{array}$$

  in $\mathbb{D}$, a square

$$\begin{array}{ccc} \mathcal{A}(A) & \xrightarrow{\mathcal{A}(f)} & \mathcal{A}(B) \\ {\scriptstyle \mathcal{A}(j)}\downarrow & \mathcal{A}(\alpha) & \downarrow{\scriptstyle \mathcal{A}(k)} \\ \mathcal{A}(C) & \xrightarrow[\mathcal{A}(g)]{} & \mathcal{A}(D) \end{array}$$

  in **Cat**.
- For any two horizontal maps $f : A \to B$ and $g : B \to E$ in $\mathbb{D}$, we have a square $\mu_{f,g} : \mathcal{A}(f) \odot \mathcal{A}(g) \to \mathcal{A}(f \mid g)$ called the *compositor*:

$$\begin{array}{ccc} \mathcal{A}(A) & \overset{\mathcal{A}(f)}{\nrightarrow} & \mathcal{A}(B) \overset{\mathcal{A}(f)}{\nrightarrow} \mathcal{A}(E) \\ \| & \mu_{f,g} & \| \\ \mathcal{A}(C) & \xrightarrow[\mathcal{A}(f \mid g)]{} & \mathcal{A}(F) \end{array} \tag{3.60}$$

This data is required to satisfy the following laws:

- (Vertical Functoriality) For vertical maps $j : D \to D'$ and $k : D' \to D''$, we have that
$$\mathcal{A}(k \circ j) = \mathcal{A}(k) \circ \mathcal{A}(j)$$
and that $\mathcal{A}(\mathrm{id}_D) = \mathrm{id}_{\mathcal{A}(D)}$.[b]
- (Horizontal Lax Functoriality) For horizontal maps $f : D_1 \to D_2$, $g : D_2 \to D_3$ and $h : D_3 \to D_4$, the compositors μ satisfy the following associativity and unitality conditions:
  - (Associativity)
$$\frac{\mu_{f,g} \mid \mathcal{A}(h)}{\mu_{g \circ f, h}} = \frac{\mathcal{A}(f) \mid \mu_{g,h}}{\mu_{f, h \circ g}}.$$

  - (Unitality) The profunctor $\mathcal{A}(\mathrm{id}_{D_1}) : \mathcal{A}(D_1) \nrightarrow \mathcal{A}(D_1)$ is the identity profunctor, $\mathcal{A}(\mathrm{id}_{D_1}) = \mathcal{A}(D_1)$. Furthermore, $\mu_{\mathrm{id}_{D_1},f}$ and $\mu_{f,\mathrm{id}_{D_2}}$ are equal to the isomorphisms of Exercise 3.51 given by the naturality of $\mathcal{A}(f)$ on the left and right respectively. We may sumarize this may saying that

$$\mu_{\mathrm{id},f} = \mathrm{id}_{\mathcal{A}(f)} = \mu_{f,\mathrm{id}}.$$

- (Naturality of Compositors) For any horizontally composable squares α and β with bottom horizontal maps f and g respectively,

$$\frac{\mathcal{A}(\alpha) \mid \mathcal{A}(\beta)}{\mu_{f,g}} = \frac{\mu_f \mid \mu_g}{\mathcal{A}(\alpha \mid \beta)}.$$

[a] This is what an expert would call a *unital (or normal) lax double functor*, but we won't need this concept in any other setting.

[b] Here, we are hiding some coherence issues. While our doubly indexed category of deterministic systems will satisfy this functoriality condition on the nose, we will soon see a doubly indexed category of matrices of sets for which this law only holds up to a coherence isomorphism. Again, the issue invovles shuffling parentheses around, and we will sweep it under the rug.

That's another big definition! It seems like it will be a slog to actually ever prove that something is a doubly indexed category. Luckily, in our cases, these proofs will go quite smoothly. This is because each of the three laws of a doubly indexed category has a sort of sister law from the definition of a double category which will help us prove it.

- The Vertical Functoriality law will often involve the vertical associativity and unitality of squares in the indexing base.
- The Horizontal Lax Functoriality law will often involve the horizontal associativity and unitality of squares in the indexing base.
- The Naturality of Compositors law will often involve the interchange law in the indexing base.

We'll see how these sisterhoods play out in practice as we define the doubly indexed categories of deterministic systems and vectors of sets.

The doubly indexed category of deterministic systems Let's show that deterministic systems do indeed form a doubly indexed category

$$\mathbf{Sys} : \mathbf{Arena} \to \mathbf{Cat}.$$

Definition 3.61. The doubly indexed category $\mathbf{Sys} : \mathbf{Arena} \to \mathbf{Cat}$ is defined as follows:

- Our indexing base is the double category **Arena** of arenas, since we will arrange our systems according to their interface.
- To every arena $\left(\begin{smallmatrix} I \\ O \end{smallmatrix}\right)$, we associate the category $\mathbf{Sys}\left(\begin{smallmatrix} I \\ O \end{smallmatrix}\right)$ of systems with interface $\left(\begin{smallmatrix} I \\ O \end{smallmatrix}\right)$ and behaviors whose chart is the identity chart on $\left(\begin{smallmatrix} I \\ O \end{smallmatrix}\right)$ (Definition 2.43).
- To every lens $\left(\begin{smallmatrix} w^\sharp \\ w \end{smallmatrix}\right) : \left(\begin{smallmatrix} I \\ O \end{smallmatrix}\right) \leftrightarrows \left(\begin{smallmatrix} I' \\ O' \end{smallmatrix}\right)$, we associate the functor $\mathbf{Sys}\left(\begin{smallmatrix} w^\sharp \\ w \end{smallmatrix}\right) : \mathbf{Sys}\left(\begin{smallmatrix} I \\ O \end{smallmatrix}\right) \to$

$\mathbf{Sys}\left(\begin{smallmatrix} I \\ O \end{smallmatrix}\right)$ given by wiring according to $\left(\begin{smallmatrix} w^\sharp \\ w \end{smallmatrix}\right)$:

$$\mathbf{Sys}\begin{pmatrix} w^\sharp \\ w \end{pmatrix}(\mathsf{S}) = \frac{\mathsf{S}}{\left(\begin{smallmatrix} w^\sharp \\ w \end{smallmatrix}\right)}.$$

- To every chart $\left(\begin{smallmatrix} f_\flat \\ f \end{smallmatrix}\right) : \left(\begin{smallmatrix} I \\ O \end{smallmatrix}\right) \rightrightarrows \left(\begin{smallmatrix} I' \\ O' \end{smallmatrix}\right)$, we associate the profunctor $\mathbf{Sys}\left(\begin{smallmatrix} f_\flat \\ f \end{smallmatrix}\right) : \mathbf{Sys}\left(\begin{smallmatrix} I \\ O \end{smallmatrix}\right) \nrightarrow \mathbf{Sys}\left(\begin{smallmatrix} I' \\ O' \end{smallmatrix}\right)$ which sends the $\left(\begin{smallmatrix} I \\ O \end{smallmatrix}\right)$-system T and the $\left(\begin{smallmatrix} I' \\ O' \end{smallmatrix}\right)$-system S to the set of behaviors $\mathsf{T} \longrightarrow \mathsf{S}$ with chart $\left(\begin{smallmatrix} f_\flat \\ f \end{smallmatrix}\right)$:

$$\mathbf{Sys}\begin{pmatrix} f_\flat \\ f \end{pmatrix}(\mathsf{T},\mathsf{S}) = \left\{ \phi : \mathsf{State}_\mathsf{T} \longrightarrow \mathsf{State}_\mathsf{S} \;\middle|\; \phi \text{ is a behavior with chart } \begin{pmatrix} f_\flat \\ f \end{pmatrix} \right\}$$

$$= \left\{ \begin{array}{ccc} & \overset{\left(\begin{smallmatrix} \phi\circ\pi_2 \\ \phi \end{smallmatrix}\right)}{\dashrightarrow} & \\ \left(\begin{smallmatrix} \mathsf{State}_\mathsf{T} \\ \mathsf{State}_\mathsf{T} \end{smallmatrix}\right) & & \left(\begin{smallmatrix} \mathsf{State}_\mathsf{S} \\ \mathsf{State}_\mathsf{S} \end{smallmatrix}\right) \\ \left(\begin{smallmatrix} \mathrm{update}_\mathsf{T} \\ \mathrm{expose}_\mathsf{T} \end{smallmatrix}\right)\downarrow\uparrow & & \downarrow\uparrow\left(\begin{smallmatrix} \mathrm{update}_\mathsf{S} \\ \mathrm{expose}_\mathsf{S} \end{smallmatrix}\right) \\ \left(\begin{smallmatrix} \mathsf{In}_\mathsf{T} \\ \mathsf{Out}_\mathsf{T} \end{smallmatrix}\right) & \underset{\left(\begin{smallmatrix} f^\sharp \\ f \end{smallmatrix}\right)}{\Longrightarrow} & \left(\begin{smallmatrix} \mathsf{In}_\mathsf{S} \\ \mathsf{Out}_\mathsf{S} \end{smallmatrix}\right) \end{array} \right\}$$

  We saw this profunctor in Example 3.54.
- To every square α, we assign the morphism of profunctors given by composing vertically with α in **Arena**:

$$\mathbf{Sys}(\alpha)(\phi) = \frac{\phi}{\alpha}.$$

  We saw in Exercise 3.58 that this was a natural transformation.
- The compositor is given by horizontal composition in the double category of arenas:

$$\mu_{\left(\begin{smallmatrix} f_\flat \\ f \end{smallmatrix}\right),\left(\begin{smallmatrix} g_\flat \\ g \end{smallmatrix}\right)} : \mathbf{Sys}\begin{pmatrix} f_\flat \\ f \end{pmatrix} \odot \mathbf{Sys}\begin{pmatrix} g_\flat \\ g \end{pmatrix} \longrightarrow \mathbf{Sys}\left(\begin{pmatrix} f_\flat \\ f \end{pmatrix} \fatsemi \begin{pmatrix} g_\flat \\ g \end{pmatrix}\right)$$

$$(\phi,\psi) \mapsto \phi \mid \psi$$

Let's check now that this does indeed satisfy the laws of a doubly indexed category. The task may appear to loom over us; there are quite a few laws, and there is a lot of data involved. But nicely, they all follow quickly from a bit of fiddling in the double category of arenas.

- (Vertical Functoriality) We show that $\mathbf{Sys}\left(\left(\begin{smallmatrix} k^\sharp \\ k \end{smallmatrix}\right) \circ \left(\begin{smallmatrix} j^\sharp \\ j \end{smallmatrix}\right)\right) = \mathbf{Sys}\left(\begin{smallmatrix} k^\sharp \\ k \end{smallmatrix}\right) \circ \mathbf{Sys}\left(\begin{smallmatrix} j^\sharp \\ j \end{smallmatrix}\right)$ by

vertical associativity:

$$\mathbf{Sys}\left(\binom{k^\sharp}{k} \circ \binom{j^\sharp}{j}\right)(\phi) = \frac{\phi}{\left(\frac{\binom{j^\sharp}{j}}{\binom{k^\sharp}{k}}\right)} = \frac{\left(\frac{\phi}{\binom{j^\sharp}{j}}\right)}{\binom{k^\sharp}{k}}$$
$$= \mathbf{Sys}\binom{k^\sharp}{k} \circ \mathbf{Sys}\binom{j^\sharp}{j}(\phi).$$

- (Horizontal Lax Functoriality) This law follows from horizontal associativity in **Arena**.

$$\mu(\mu(\phi,\psi),\xi) = (\phi \mid \psi) \mid \xi = \phi \mid (\psi \mid \xi) = \mu(\phi,\mu(\psi,\xi)). \tag{3.62}$$

- (Naturality of Compositor) This law follows from interchange in **Arena**.

$$\left(\frac{\mathbf{Sys}(\alpha) \mid \mathbf{Sys}(\beta)}{\mu}\right)(\phi,\psi) = \frac{\phi}{\alpha}\middle|\frac{\psi}{\beta} = \frac{\phi \mid \psi}{\alpha \mid \beta}$$
$$= \left(\frac{\mu}{\mathbf{Sys}(\alpha \mid \beta)}\right)(\phi,\psi).$$

The doubly indexed category of vectors of sets In addition to our doubly indexed category of systems, we have a doubly indexed category of "vectors of sets".

Classically, an $m \times n$ matrix M can act on a vector v of length n by multiplication to get another vector Mv of length m. We can generalize this to matrices of sets if we define a vector of sets of length A to be a dependent set $V : A \to \mathbf{Set}$.

Definition 3.63. For a set A, we define the category of *vectors of sets of length* A to be

$$\mathbf{Vec}(A) \coloneqq \mathbf{Set}^A$$

the category of sets depending on A.

Given a $(B \times A)$-matrix M, we can treat a A-vector V as a $A \times 1$ matrix and form the $B \times 1$ matrix MV. This gives us a functor

$$\begin{aligned}\mathbf{Vec}(M) : \mathbf{Vec}(A) &\to \mathbf{Vec}(B)\\ V &\mapsto (MV)_b = \sum_{a \in A} M_{ba} \times V_a\\ f : V \to W &\mapsto ((a,m,v) \mapsto (a,m,f(v)))\end{aligned}$$

which we refer to as the linear functor given by M.

Definition 3.64. The doubly indexed category $\mathbf{Vec} : \mathbf{Matrix} \to \mathbf{Cat}$ of vectors of sets is defined by:

- Its indexing base is the double category of matrices of sets.
- To every set A, we assign the category $\mathbf{Vec}(A) = \mathbf{Set}^A$ of vectors of length A.
- To every $(B \times A)$-matrix $M : A \to B$, we assign the linear functor $\mathbf{Vec}(M) : \mathbf{Vec}(A) \to \mathbf{Vec}(B)$ given by M (Definition 3.63).
- To every function $f : A \to B$, we associate the profunctor $\mathbf{Vec}(f) : \mathbf{Vec}(A) \nrightarrow \mathbf{Vec}(B)$ defined by
$$\mathbf{Vec}(f)(V, W) = \{F : (a \in A) \to V_a \to W_{f(a)}\}.$$
That is, $F \in \mathbf{Vec}(f)(V, W)$ is a family of functions $F(a, -) : V_a \to W_{f(a)}$ indexed by $a \in A$. This is natural by index-wise composition.
- To every square
$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle M}\downarrow & \alpha & \downarrow{\scriptstyle N} \\ C & \xrightarrow[g]{} & D \end{array}$$
that is, family of functions $\alpha_{ca} : M_{ca} \to N_{g(c)f(a)}$, we associate the square
$$\begin{array}{ccc} \mathbf{Vec}(A) & \xrightarrow{\mathbf{Vec}(f)} & \mathbf{Vec}(B) \\ {\scriptstyle \mathbf{Vec}(M)}\downarrow & \mathbf{Vec}(\alpha) & \downarrow{\scriptstyle \mathbf{Vec}(N)} \\ \mathbf{Vec}(C) & \xrightarrow[\mathbf{Vec}(g)]{} & \mathbf{Vec}(D) \end{array}$$
defined by sending a family of functions $F : (a \in A) \to V_a \to W_{f(a)}$ in $\mathbf{Vec}(f)(V, W)$ to the family
$$\mathbf{Vec}(\alpha)(F) : (c \in C) \to MV_c \to MW_{g(c)}$$
$$\mathbf{Vec}(\alpha)(F)(c, (a, m, v)) = (f(a), \alpha(m), F(a, v))$$
That is, $\mathbf{Vec}(\alpha)(F)(c, -)$ takes an element $(a, m, v) \in MV_c = \sum_{a \in A} M_{ca} \times V_a$ and gives the elements $(f(a), \alpha(m), F(a, v))$ of $MW_{g(c)} = \sum_{b \in B} N_{g(c)b} \times W_b$.
- The compositor is given by componentwise composition: If $f : A \to B$ and $g : B \to C$ and $F \in \mathbf{Vec}(f)(V, W)$ and $G \in \mathbf{Vec}(g)(W, U)$, then
$$\mu_{f,g}(F, G) : (a \in A) \to V_a \to U_{gf(a)}$$
$$\mu_{f,g}(F, G)(a, v) \coloneqq G(f(a), F(a, v)).$$

It might seem like it will turn out to be a big hassle to show that this definition satisfies all the laws of a doubly indexed category. Like with the doubly indexed category of arenas, we will find that all the laws follow for matrices by fiddling around in the double category of matrices.

Let's first rephrase the above definition in terms of the category of matrices. We note that a vector of sets $V \in \mathbf{Vec}(A)$ is equivalently a matrix $V : 1 \to A$. Then the linear functor $\mathbf{Vec}(M) : \mathbf{Vec}(A) \to \mathbf{Vec}(B)$ is given by matrix multiplication, or in double category notation:

$$\mathbf{Vec}(M)(V) = \frac{V}{M}.$$

This means that the Vertical Functoriality law follows by vertical associativity in the double category of matrices, which is to say associativity of matrix multiplication.

Similarly, we can interpret the profunctor $\mathbf{Vec}(f)$ for $f : A \to B$ in terms of the double category **Matrix**. An element $F \in \mathbf{Vec}(f)(V, W)$ is equivalently a square of the following form in **Matrix**:

$$\begin{array}{ccc} 1 & = & 1 \\ {\scriptstyle V}\downarrow & F & \downarrow{\scriptstyle W} \\ A & \xrightarrow[f]{} & B \end{array}$$

Therefore, we can describe $\mathbf{Vec}(f)(V, W)$ as the following set:

$$\mathbf{Vec}(f)(V, W) = \left\{ F \;\middle|\; \begin{array}{ccc} 1 & = & 1 \\ {\scriptstyle V}\downarrow & F & \downarrow{\scriptstyle W} \\ A & \xrightarrow[f]{} & B \end{array} \right\}$$

Then the Horizontal Lax Functoriality laws follow from associativity and unitality of horizontal composition of squares in **Matrix**!

Finally, we need to interpret the rather fiddly transformation $\mathbf{Vec}(\alpha)$ in terms of the double category of matrices. Its a matter of unfolding the definitions to see that $\mathbf{Vec}(\alpha)(F) = \frac{F}{\alpha}$ in **Matrix**, and therefore that the Naturality of Compositors law follows by the interchange law.

Remark 3.65. If this argument seemed wholly too similar to the one we gave for the doubly indexed category of systems, your suspicions are not misplaced. We will see in Chapter 2 that both are instances of a very general *vertical slice construction*.

3.7 The big theorem: representable doubly indexed functors

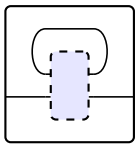
We have now introduced all the characters in our play: the double categories of arenas and matrices, and doubly indexed categories of systems and vectors. In this section, we will put the plot in motion.

In Section 3.2, we saw that the steady states of dynamical systems with interface $\binom{I}{O}$ compose like an $I \times O$ matrix. We proved a few propositions to this effect, namely Proposition 3.12 and Proposition 3.14, but we didn't precisely mark out the scope of these results, or describe the full range of laws that are satisfied.

In this section, we will generalize the results of that section to *all behaviors* of systems, not just steady states. We will precisely state all the ways that behaviors can be composed by systems, and we will give a condition on the kinds of behaviors for which we can calculate the behavior of a wired together system entirely from the behavior of its component systems. All of this will be organized into a *doubly indexed functor* $\text{Behave}_{\mathsf{T}} : \mathbf{Sys} \to \mathbf{Vec}$ which will send a system S to its set of T-shaped behaviors.

3.7.1 Turning lenses into matrices: Double Functors

In Section 3.2, we saw how we could re-interpret a wiring diagram as a schematic for multiplying, tensoring, and tracing matrices. At the very end, in Proposition 3.16, we saw that we can take the trace $\text{tr}_A M$ of a $(A \times C) \times (A \times B)$-matrix M by considering it as a $(A \times C) \times (B \times C)$ length vector and then multiplying it by a big but very sparse $(C \times B) \times ((A \times C) \times (B \times C))$-matrix Tr^A. Taking the trace of a matrix corresponded to the wiring diagram



In this section, we will see a general formula for taking an arbitrary lens and turning it into a matrix. Mutliplying by the matrix will then correspond to wiring according to that lens.

This process of turning a lens into a matrix will give us a functor **Lens** $\to$ **Matrix** from the category of lenses to the category of matrices of sets.

The resulting matrices will have entries that are either 1 or $\emptyset$; we can think of this as telling us whether (1) or not ($\emptyset$) the two charts are to be wired together. As we saw in Example 2.61, we can see a square in the double category of arenas as telling us whether how a chart can be wired together along a lens into another chart. Therefore, we will take the entries of our matrices to be the sets of appropriate squares in arena — but there is either a single square (if the appropriate equations hold) or no square (if they don't), so we will end up with a matrix whose entries either have a single element or are empty.

Proposition 3.66. For any arena $\binom{I}{O}$, there is a functor

$$\mathbf{Chart}\left(\binom{I}{O}, -\right) : \mathbf{Lens} \to \mathbf{Matrix}$$

from the category of lenses to the category of matrices of sets which sends an arena $\binom{A^-}{A^+}$ to the set $\mathbf{Chart}\left(\binom{I}{O}, \binom{A^-}{A^+}\right)$ of charts from $\binom{I}{O}$ to $\binom{A^-}{A^+}$, and which sends a lens $\binom{w^\sharp}{w} : \binom{A^-}{A^+} \leftrightarrows \binom{B^-}{B^+}$ to the $\mathbf{Chart}\left(\binom{I}{O}, \binom{B^-}{B^+}\right) \times \mathbf{Chart}\left(\binom{I}{O}, \binom{A^-}{A}\right)$ matrix of sets

$$\mathbf{Chart}\left(\binom{I}{O}, \binom{w^\sharp}{w}\right) : \mathbf{Chart}\left(\binom{I}{O}, \binom{B^-}{B^+}\right) \times \mathbf{Chart}\left(\binom{I}{O}, \binom{A^-}{A}\right) \to \mathbf{Set}$$

$$\left(\binom{f_\flat}{f}, \binom{g_\flat}{g}\right) \mapsto \left\{ \text{The set of squares } \begin{array}{ccc} \binom{I}{O} & \xrightarrow{\binom{f_\flat}{f}} & \binom{A^-}{A^+} \\ \| & & \downarrow\uparrow \binom{w^\sharp}{w} \\ \binom{I}{O} & \xrightarrow[\binom{g^\sharp}{g}]{} & \binom{B^-}{B^+} \end{array} \text{ in } \mathbf{Arena} \right\}$$

$$= \begin{cases} 1 & \text{if } \begin{cases} g(o) = w(f(o)) & \text{for all } o \in O, \\ f_\flat(i,o) = w^\sharp(f(o), g_\flat(i,o)) & \text{for all } i \in I \text{ and } o \in O. \end{cases} \\ \emptyset & \text{otherwise} \end{cases}$$

Proof. By vertical composition of squares,

$$\begin{array}{ccc} \binom{I}{O} & \xrightarrow{\binom{f_\flat}{f}} & \binom{A^-}{A^+} \\ \| & & \downarrow\uparrow \binom{w^\sharp}{w} \\ \binom{I}{O} & \xrightarrow[\binom{g_\flat}{g}]{} & \binom{B^-}{B^+} \\ \| & & \downarrow\uparrow \binom{v^\sharp}{v} \\ \binom{I}{O} & \xrightarrow[\binom{h_\flat}{h}]{} & \binom{I'}{O'} \end{array} = \begin{array}{ccc} \binom{I}{O} & \xrightarrow{\binom{f_\flat}{f}} & \binom{A^-}{A^+} \\ \| & & \downarrow\uparrow \binom{w^\sharp}{w} \fatsemi \binom{v^\sharp}{v} \\ \binom{I}{O} & \xrightarrow[\binom{h_\flat}{h}]{} & \binom{I}{O'} \end{array}$$

there is always a map from the composite of two of these matrices to the matrix

described by the composite. It is not, however, obvious that this map is a bijection — which is what we need to prove functoriality.

Suppose we have a square as on the left hand side; let's see that we can factor it into two squares as on the right hand side. We need to construct the middle chart $\binom{g_\flat}{g} : \binom{I}{O} \rightrightarrows \binom{B^-}{B^+}$ from $\binom{f_\flat}{f}$ and $\binom{h_\flat}{h}$. For the bottom of top square to commute, we see that g must equal $w \circ f$, so we can define $g \coloneqq w \circ f$. On the other hand, for the top of the bottom square to commute, we must have that $g_\flat(i,o) = v^\sharp(g(o), h_\flat(i,o))$; again, we can take this as a definition. It remains to show that the other half of each square commutes. For the top of the top square to commute means that

$$f_\flat(i,o) = w^\sharp(f(o), g_\flat(i,o))$$

which we can see holds by

$$\begin{aligned} w^\sharp(f(o), g_\flat(i,o)) &= w^\sharp(f(o), v^\sharp(g(o), h_\flat(i,o))) \\ &= w^\sharp(f(o), v^\sharp(wf(o), h_\flat(i,o))) \\ &= f_\flat(i,o) && \text{by the commutativity of the square on the right.} \end{aligned}$$

On the other hand, to show that the bottom of the bottoms square commutes, we need that $h = v \circ g$. But by hypothesis, $h = v \circ w \circ f$, and we defined $g = w \circ f$. □

Example 3.67. Let's see what happens when we take the functor $\mathbf{Chart}\left(\binom{I}{O}, -\right)$ for the arena $\binom{1}{1}$. A chart $\binom{a^-}{a^+} : \binom{1}{1} \rightrightarrows \binom{A^-}{A^+}$ is just a pair of elements $a^- \in A^-$ and $a^+ \in A^+$, so

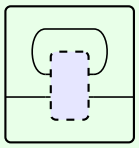
$$\mathbf{Chart}\left(\binom{1}{1}, \binom{A^-}{A^+}\right) = A^- \times A^+.$$

Now, if we have a lens $\binom{w^\sharp}{w} : \binom{A^-}{A^+} \leftrightarrows \binom{B^-}{B^+}$, we have a square

$$\begin{array}{ccc} \binom{1}{1} & \xRightarrow[\binom{b^-}{b^+}]{\binom{a^-}{a^+}} & \binom{A^-}{A^+} \\ \| & & \downarrow\uparrow \binom{w^\sharp}{w} \\ \binom{1}{1} & \xRightarrow[\binom{b^-}{b^+}]{\binom{a^-}{a^+}} & \binom{B^-}{B^+} \end{array}$$

if and only if $w(a^+) = b^+$ and $w^\sharp(a^+, b^-) = a^-$. Thinking of $\binom{w^\sharp}{w}$ as a wiring diagram, this would mean that b^+ is that part of a^+ which is passed forward on the outgoing wires, and a^- is the inner input which comes from the inner output a^+ and outer input b^-.

To take a concrete example, suppose that $\binom{w^\sharp}{w}$ were the following wiring diagarm:



That is, let's take $A^+ = X \times Y$ and $A^- = X \times Z$, and $B^+ = Y$ and $B^- = Z$, and

$$w(x,y) = y$$
$$w^\sharp((x,y),z) = (x,z).$$

Using the definition above, we can calculate the resulting matrix $\mathbf{Chart}\left(\binom{I}{O}, \binom{w^\sharp}{w}\right)$ as having $(((x,y),(x',z)),(y',z'))$-entry

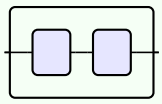
$$\begin{cases} 1 & \text{if } w(x,y) = y' \text{ and } w^\sharp((x,y),z) = (x',z') \\ \emptyset & \text{otherwise.} \end{cases}$$

or, by the definition of $\binom{w^\sharp}{w}$,

$$\begin{cases} 1 & \text{if } x = x', y = y', \text{ and } z = z' \\ \emptyset & \text{otherwise.} \end{cases}$$

which was the definition of Tr^X given in Proposition 3.16!

Exercise 3.68. Let $\binom{w^\sharp}{w} : \binom{A\times B}{B\times C} \leftrightarrows \binom{A}{C}$ be the wiring diagram



Calculate the entries of the matrix $\mathbf{Chart}\left(\binom{1}{1}, \binom{w^\sharp}{w}\right)$. ◊

By the functoriality of Proposition 3.66, we can calculate the matrix of a big wiring diagram by expressing it in terms of a series of traces, and mutliplying the resulting matrices together. This means that the process of multiplying, tensoring, and tracing matrices described by a wiring diagram is well described by the matrix we constructed in Proposition 3.66, since we already know that it interprets the basic wiring diagrams correctly.

But we are also interested in charts, since we have to chart out our behaviors. So we will give a *double functor* **Arena** $\to$ **Matrix** that tells us not only how to turn a lens into a matrix, but also how this operation interacts with charts.

A double functor is, unsurprisingly, a functor between double categories. Just as a double category has a bit more than twice the information involved in a category, a double functor has a bit more than twice the information involved in a functor.

Definition 3.69. Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be double categories. A *double functor* $\mathrm{F} : \mathcal{D}_1 \to \mathcal{D}_2$ consists of:

- An object assignment $F : \mathrm{Ob}\,\mathcal{D}_1 \to \mathrm{Ob}\,\mathcal{D}_2$ which assigns an object FD in $\mathcal{D}_2$ to each object D in $\mathcal{D}_1$.
- A vertical functor $F : v\mathcal{D}_1 \to v\mathcal{D}_2$ on the vertical categories, which acts the same as the object assignment on objects.
- A horizontal functor $F : h\mathcal{D}_1 \to h\mathcal{D}_2$ on the horizontal categories, which acts the same as the object assignment on objects.
- For every square

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle j}\downarrow & \alpha & \downarrow{\scriptstyle k} \\ C & \xrightarrow[g]{} & D \end{array}$$

  in $\mathcal{D}_1$, a square

$$\begin{array}{ccc} FA & \xrightarrow{Ff} & FB \\ {\scriptstyle Fj}\downarrow & F\alpha & \downarrow{\scriptstyle Fk} \\ FC & \xrightarrow[Fg]{} & FD \end{array}$$

  such that the following laws hold:
  - F commutes with horizontal compostition: $F(\alpha \mid \beta) = F\alpha \mid F\beta$.
  - F commutes with vertical comopsition: $F\left(\frac{\alpha}{\beta}\right) = \frac{F\alpha}{F\beta}$.
  - F sends horizontal identities to horizontal identities, and vertical identities to vertical identities.

We are now ready to define the double functor $\mathbf{Chart}\left(\binom{I}{O}, -\right) : \mathbf{Arena} \to \mathbf{Matrix}$ represented by an arena $\binom{I}{O}$.

Proposition 3.70. There is a double functor

$$\mathbf{Chart}\left(\begin{pmatrix} I \\ O \end{pmatrix}, -\right) : \mathbf{Chart} \to \mathbf{Matrix}$$

which acts in the following way:

- An arena $\begin{pmatrix} A^- \\ A^+ \end{pmatrix}$ gets sent to the set $\mathbf{Chart}\left(\begin{pmatrix} I \\ O \end{pmatrix}, \begin{pmatrix} A^- \\ A^+ \end{pmatrix}\right)$ of charts from $\begin{pmatrix} I \\ O \end{pmatrix}$ to $\begin{pmatrix} A^- \\ A^+ \end{pmatrix}$.
- The vertical functor is $\mathbf{Chart}\left(\begin{pmatrix} I \\ O \end{pmatrix}, -\right) : \mathbf{Lens} \to \mathbf{Matrix}$ from Proposition 3.66.
- The horizontal functor is the representable functor $\mathbf{Chart}\left(\begin{pmatrix} I \\ O \end{pmatrix}, -\right) : \mathbf{Arena} \to \mathbf{Set}$ which acts on a chart $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} A^- \\ A^+ \end{pmatrix} \rightrightarrows \begin{pmatrix} B^- \\ B^+ \end{pmatrix}$ by post-composition.
- To a square

$$\alpha = \begin{array}{ccc} \begin{pmatrix} A^- \\ A^+ \end{pmatrix} & \overset{\begin{pmatrix} f_\flat \\ f \end{pmatrix}}{\Longrightarrow} & \begin{pmatrix} B^- \\ B^+ \end{pmatrix} \\ \begin{pmatrix} j^\sharp \\ j \end{pmatrix}\downarrow\uparrow & & \downarrow\uparrow\begin{pmatrix} k^\sharp \\ k \end{pmatrix} \\ \begin{pmatrix} C^- \\ C^+ \end{pmatrix} & \underset{\begin{pmatrix} g^\sharp \\ g \end{pmatrix}}{\Longrightarrow} & \begin{pmatrix} D^- \\ D^+ \end{pmatrix} \end{array}$$

  in the double category of arenas, we give the square

$$\mathbf{Chart}\left(\begin{pmatrix} I \\ O \end{pmatrix}, \alpha\right) = \begin{array}{ccc} \mathbf{Chart}\left(\begin{pmatrix} I \\ O \end{pmatrix}, \begin{pmatrix} A^- \\ A^+ \end{pmatrix}\right) & \xrightarrow{\mathbf{Chart}\left(\begin{pmatrix} I \\ O \end{pmatrix}, \begin{pmatrix} f_\flat \\ f \end{pmatrix}\right)} & \mathbf{Chart}\left(\begin{pmatrix} I \\ O \end{pmatrix}, \begin{pmatrix} B^- \\ B^+ \end{pmatrix}\right) \\ {\scriptstyle \mathbf{Chart}\left(\begin{pmatrix} I \\ O \end{pmatrix}, \begin{pmatrix} j^\sharp \\ j \end{pmatrix}\right)}\downarrow & & \downarrow{\scriptstyle \mathbf{Chart}\left(\begin{pmatrix} I \\ O \end{pmatrix}, \begin{pmatrix} k^\sharp \\ k \end{pmatrix}\right)} \\ \mathbf{Chart}\left(\begin{pmatrix} I \\ O \end{pmatrix}, \begin{pmatrix} C^- \\ C^+ \end{pmatrix}\right) & \xrightarrow[\mathbf{Chart}\left(\begin{pmatrix} I \\ O \end{pmatrix}, \begin{pmatrix} g^\sharp \\ g \end{pmatrix}\right)]{} & \mathbf{Chart}\left(\begin{pmatrix} I \\ O \end{pmatrix}, \begin{pmatrix} D^- \\ D^+ \end{pmatrix}\right) \end{array}$$

  in the double category of matrices defined by horizontal composition of squares in **Arena** (remember that the entries of these matrices are sets of squares in **Arena**, even though that means they either have a single element or no elements).

$$\mathbf{Chart}\left(\begin{pmatrix} I \\ O \end{pmatrix}, \alpha\right)(\beta) = \beta \mid \alpha.$$

Proof. We can write the double functor $\mathbf{Chart}\left(\begin{pmatrix} I \\ O \end{pmatrix}, -\right)$ entirely in terms of the double category **Arena**:

- It sends an arena $\begin{pmatrix} A^- \\ A^+ \end{pmatrix}$ to the set of charts (horizontal maps) $\begin{pmatrix} f_\flat \\ f \end{pmatrix} : \begin{pmatrix} I \\ O \end{pmatrix} \rightrightarrows \begin{pmatrix} A^- \\ A^+ \end{pmatrix}$.
- It sends a chart $\begin{pmatrix} g_\flat \\ g \end{pmatrix}$ to the map $\begin{pmatrix} f_\flat \\ f \end{pmatrix} \mapsto \begin{pmatrix} f_\flat \\ f \end{pmatrix}\middle|\begin{pmatrix} g_\flat \\ g \end{pmatrix}$.
- It sends a lens $\begin{pmatrix} w^\sharp \\ w \end{pmatrix}$ to the set of squares $\beta : \begin{pmatrix} I \\ O \end{pmatrix} \to \begin{pmatrix} w^\sharp \\ w \end{pmatrix}$, indexed by their top and bottom boundaries.
- It sends a square α to the map given by horizontal compostion $\beta \mapsto \beta \mid \alpha$.

We can see that this double functor (let's call it F, for short) takes seriously the idea

that "squares are charts between lenses" from Example 2.61. From this description, and the functoriality of Proposition 3.66, we can see that the assignments above satisfy the double functor laws.

- Horizontal functoriality follows from horizontal associativity in **Arena**:

$$F(\alpha \mid \beta)(\gamma) = \gamma \mid (\alpha \mid \beta) = (\gamma \mid \alpha) \mid \beta = F(\alpha) \mid F(\beta)(\gamma).$$

- Vertical functoriality follows straight from the definitions:

$$F\left(\frac{\alpha}{\beta}\right)(\_, \gamma, \delta) = (\_, \gamma \mid \alpha, \delta \mid \beta) = \frac{F(\alpha)(\gamma)}{F(\beta)(\delta)}.$$

- It's pretty straightforward to check that identities get sent to identities.

□

3.7.2 How behaviors of systems wire together: doubly indexed functors

We now come to the mountaintop. It's been quite a climb, and we're almost there.

We can now describe all the ways that behaviors of systems get put together when we wire systems together. There are a bunch of laws governing how behaviors get put together, and we organize them all into the notion of a *lax doubly indexed functor*. To any system T, we will give a lax doubly indexed functor

$$\text{Behave}_{\mathsf{T}} : \mathbf{Sys} \to \mathbf{Vec}.$$

Here's what that means.

Definition 3.71. Let $\mathcal{A} : \mathcal{D}_1 \to \mathbf{Cat}$ and $\mathcal{B} : \mathcal{D}_2 \to \mathbf{Cat}$ be doubly indexed categories. A *lax doubly indexed functor* $(F^0, F) : \mathcal{A} \nrightarrow \mathcal{B}$ consists of:

- A double functor

$$F^0 : \mathcal{D}_1 \to \mathcal{D}_2.$$

- For each object $D \in \mathcal{D}_1$, a functor

$$F^D : \mathcal{A}(D) \to \mathcal{B}(F^0 D).$$

- For every vertical map $j : D \to D'$ in $\mathcal{D}_1$, a natural transformation

$$F^j : \mathcal{B}(F^0 j) \circ F^D \to F^{D'} \circ \mathcal{A}(j).$$

  We ask that $F^{\mathrm{id}_D} = \mathrm{id}$.

- For every horizontal map $f : D \to D'$, a square

$$\begin{array}{ccc} \mathcal{A}(D) & \xrightarrow{\mathcal{A}(f)} & \mathcal{A}(D') \\ {\scriptstyle F^D}\downarrow & F^j & \downarrow{\scriptstyle F^{D'}} \\ \mathcal{B}(F^0 D) & \xrightarrow[\mathcal{B}(F^0 f)]{} & \mathcal{B}(F^0 D') \end{array}$$

in **Cat**. We ask that $F^{\mathrm{id}_D} = \mathrm{id}$.

This data is required to satisfy the following laws:

- (Vertical Lax Functoriality) For composable vertical arrows $j : D \to D'$ and $k : D' \to D''$,
$$F^{\frac{j}{k}} = (F^k \mathcal{A}(j)) \circ (\mathcal{B}(k) F^j).$$
- (Horizontal functoriality) For composable horizontal arrows $f : D \to D'$ and $g : D' \to D''$,
$$\frac{\mu^{\mathcal{A}}_{f,g}}{F^{f|g}} = \frac{F^f \mid F^g}{\mu^{\mathcal{B}}_{F^0 f, F^0 g}}.$$
- (Functorial Interchange) For any square
$$\begin{array}{ccc} D_1 & \xrightarrow{f} & D_2 \\ {\scriptstyle j}\downarrow & \alpha & \downarrow{\scriptstyle k} \\ D_3 & \xrightarrow[g]{} & D_4 \end{array}$$
in $\mathcal{D}_1$, we have that
$$F^j \left| \frac{\mathcal{A}(\alpha)}{F^g} = \frac{F^f}{\mathcal{B}(\alpha)} \right| F^k .$$

Theorem 3.72. Let T be a deterministic system. There is a lax doubly indexed functor $\mathrm{Behave}_{\mathsf{T}} : \mathbf{Sys} \to \mathbf{Vec}$ which sends systems to their sets of T-shaped behaviors.

Let's see what this theorem is really asking for while we construct it. As with many of the constructions we have been seeing, the hard part is understanding what we are supposed to be constructing; once we do that, the answer will always be "compose in the appropriate way in the appropriate double category".

- First, we need $\mathrm{Behave}^0_{\mathsf{T}} : \mathbf{Arena} \to \mathbf{Matrix}$ which send an arena to the set of charts from $\binom{\mathsf{In}_{\mathsf{T}}}{\mathsf{Out}_{\mathsf{T}}}$ to that arena. It will send a chart to the function given by composing with that chart, and it will send a lens to a matrix that describes the wiring pattern in the lens. We've seen how to do this in Proposition 3.70:
$$\mathrm{Behave}^0_{\mathsf{T}} = \mathbf{Chart}\left(\binom{\mathsf{In}_{\mathsf{T}}}{\mathsf{Out}_{\mathsf{T}}}, -\right)$$
This is the blueprint for how our systems will compose.
- Next, for any arena $\binom{I}{O}$, we need a functor
$$\mathrm{Behave}^{\binom{I}{O}}_{\mathsf{T}} : \mathbf{Sys}\binom{I}{O} \to \mathbf{Vec}\left(\mathbf{Chart}\left(\binom{\mathsf{In}_{\mathsf{T}}}{\mathsf{Out}_{\mathsf{T}}}, \binom{I}{O}\right)\right)$$

which will send a system S with interface $\binom{I}{O}$ to its set of behaviors of shape T, indexed by their chart. That is, we make the following definition:

$$\mathrm{Behave}_{\mathsf{T}}^{\binom{I}{O}}(\mathsf{S})_{\binom{f_\flat}{f}} := \mathbf{Sys}\left(\tbinom{f_\flat}{f}\right)(\mathsf{T},\mathsf{S}).$$

This is quickly shown to be functorial by horizontal associativity of squares in **Arena**.

- For any lens $\binom{w^\sharp}{w} : \binom{I}{O} \leftrightarrows \binom{I'}{O'}$, we need a natural transformation

$$\begin{array}{ccc}
\mathbf{Sys}\binom{I}{O} & \xrightarrow{\mathrm{Behave}_{\mathsf{T}}^{\binom{I}{O}}} & \mathbf{Vec}\left(\mathbf{Chart}\left(\binom{\mathsf{In}_{\mathsf{T}}}{\mathsf{Out}_{\mathsf{T}}}, \binom{I}{O}\right)\right) \\
{\scriptstyle \mathbf{Sys}\binom{w^\sharp}{w}}\downarrow & \swarrow_{\mathrm{Behave}_{\mathsf{T}}^{\binom{w^\sharp}{w}}} & \downarrow{\scriptstyle \mathbf{Vec}\left(\mathbf{Chart}\left(\binom{\mathsf{In}_{\mathsf{T}}}{\mathsf{Out}_{\mathsf{T}}}, \binom{w^\sharp}{w}\right)\right)} \\
\mathbf{Sys}\binom{I'}{O'} & \xrightarrow[\mathrm{Behave}_{\mathsf{T}}^{\binom{B^-}{B^+}}]{} & \mathbf{Vec}\left(\mathbf{Chart}\left(\binom{\mathsf{In}_{\mathsf{T}}}{\mathsf{Out}_{\mathsf{T}}}, \binom{I}{O}\right)\right)
\end{array}$$

This will take any behaviors of component systems whose charts compatible according to the wiring pattern of $\binom{w^\sharp}{w}$ and wire them together into a behavior of the wired together systems. In other words, this will be given by vertical composition of squares in **Arena**. To see how that works, we need follow a $\binom{I}{O}$-system S around this diagram and see how this natural transformation can be described so simply. Following S around the top path of the diagram gives us the following vector of sets, we first send S to the vector of sets

$$\begin{aligned}
\tbinom{f_\flat}{f} : \binom{\mathsf{In}_{\mathsf{T}}}{\mathsf{Out}_{\mathsf{T}}} \rightrightarrows \binom{I}{O} &\mapsto \mathbf{Sys}\left(\tbinom{f_\flat}{f}\right)(\mathsf{T},\mathsf{S}) \\
&= \left\{ \begin{array}{ccc}
\binom{\mathsf{State}_{\mathsf{T}}}{\mathsf{State}_{\mathsf{T}}} & \overset{\binom{\phi\circ\pi_2}{\phi}}{\dashrightarrow} & \binom{\mathsf{State}_{\mathsf{S}}}{\mathsf{State}_{\mathsf{S}}} \\
\binom{\mathrm{update}_{\mathsf{T}}}{\mathrm{expose}_{\mathsf{T}}}\downarrow\uparrow & & \downarrow\uparrow\binom{\mathrm{update}_{\mathsf{S}}}{\mathrm{expose}_{\mathsf{S}}} \\
\binom{\mathsf{In}_{\mathsf{T}}}{\mathsf{Out}_{\mathsf{T}}} & \underset{\binom{f^\sharp}{f}}{\rightrightarrows} & \binom{I}{O}
\end{array} \right\}
\end{aligned}$$

We then multiply this by the matrix $\mathbf{Chart}\left(\begin{pmatrix}\mathsf{In}_\mathsf{T}\\ \mathsf{Out}_\mathsf{T}\end{pmatrix}, \begin{pmatrix} w^\sharp \\ w\end{pmatrix}\right)$ to get the vector of sets whose entries are pairs of the following form:

$$\begin{pmatrix} g_\flat \\ g\end{pmatrix} : \begin{pmatrix}\mathsf{In}_\mathsf{T}\\ \mathsf{Out}_\mathsf{T}\end{pmatrix} \rightrightarrows \begin{pmatrix} I' \\ O'\end{pmatrix} \mapsto \left\{ \begin{array}{cc} \begin{pmatrix}\mathsf{In}_\mathsf{T}\\ \mathsf{Out}_\mathsf{T}\end{pmatrix} \overset{\begin{pmatrix} f_\flat \\ f\end{pmatrix}}{\dashrightarrow} \begin{pmatrix} I \\ O\end{pmatrix} & \begin{pmatrix}\mathsf{State}_\mathsf{T}\\ \mathsf{State}_\mathsf{T}\end{pmatrix} \overset{\begin{pmatrix} \phi\circ\pi_2 \\ \phi\end{pmatrix}}{\dashrightarrow} \begin{pmatrix}\mathsf{State}_\mathsf{S}\\ \mathsf{State}_\mathsf{S}\end{pmatrix} \\ \Vert\Vert \quad \downarrow\uparrow \begin{pmatrix} w^\sharp \\ w\end{pmatrix}, & \begin{pmatrix}\mathsf{update}_\mathsf{T}\\ \mathsf{expose}_\mathsf{T}\end{pmatrix}\downarrow\uparrow \quad \downarrow\uparrow\begin{pmatrix}\mathsf{update}_\mathsf{S}\\ \mathsf{expose}_\mathsf{S}\end{pmatrix} \\ \begin{pmatrix}\mathsf{In}_\mathsf{T}\\ \mathsf{Out}_\mathsf{T}\end{pmatrix} \underset{\begin{pmatrix} g_\flat \\ g\end{pmatrix}}{\Longrightarrow} \begin{pmatrix} I' \\ O'\end{pmatrix} & \begin{pmatrix}\mathsf{In}_\mathsf{T}\\ \mathsf{Out}_\mathsf{T}\end{pmatrix} \underset{\begin{pmatrix} f_\flat \\ f\end{pmatrix}}{\Longrightarrow} \begin{pmatrix} I \\ O\end{pmatrix} \end{array} \right\}$$

On the other hand, following S along the bottom path has us first composing it vertically with $\begin{pmatrix} w^\sharp \\ w\end{pmatrix}$ and then finding the behaviors in it:

$$\begin{pmatrix} g_\flat \\ g\end{pmatrix} : \begin{pmatrix}\mathsf{In}_\mathsf{T}\\ \mathsf{Out}_\mathsf{T}\end{pmatrix} \rightrightarrows \begin{pmatrix} I' \\ O'\end{pmatrix} \mapsto \left\{ \begin{array}{c} \begin{pmatrix}\mathsf{State}_\mathsf{T}\\ \mathsf{State}_\mathsf{T}\end{pmatrix} \overset{\begin{pmatrix} \phi\circ\pi_2 \\ \phi\end{pmatrix}}{\dashrightarrow} \begin{pmatrix}\mathsf{State}_\mathsf{S}\\ \mathsf{State}_\mathsf{S}\end{pmatrix} \\ \begin{pmatrix}\mathsf{update}_\mathsf{T}\\ \mathsf{expose}_\mathsf{T}\end{pmatrix}\downarrow\uparrow \quad \downarrow\uparrow\begin{pmatrix}\mathsf{update}_\mathsf{S}\\ \mathsf{expose}_\mathsf{S}\end{pmatrix} \fatsemi \begin{pmatrix} w^\sharp \\ w\end{pmatrix} \\ \begin{pmatrix}\mathsf{In}_\mathsf{T}\\ \mathsf{Out}_\mathsf{T}\end{pmatrix} \underset{\begin{pmatrix} g^\sharp \\ g\end{pmatrix}}{\Longrightarrow} \begin{pmatrix} I' \\ O'\end{pmatrix} \end{array} \right\}$$

Finally, we are ready to define our natural transformation from the virst vector of sets to the second using vertical composition:

$$\mathrm{Behave}_\mathsf{T}^{\begin{pmatrix} w^\sharp \\ w\end{pmatrix}}(\mathsf{S})_{\begin{pmatrix} g_\flat \\ g\end{pmatrix}}(\square_w, \phi) = \frac{\phi}{\square_w}.$$

That this is natural for behaviors $\psi : \mathsf{S} \to \mathsf{U}$ in $\mathbf{Sys}\left(\begin{smallmatrix} I \\ O\end{smallmatrix}\right)$ follows quickly from the horizontal identity and interchange laws in **Arena**:

$$\begin{aligned} \frac{\phi \mid \psi}{\square_w} &= \frac{\phi \mid \psi}{\square_w \mid \begin{pmatrix} w^\sharp \\ w\end{pmatrix}} \\ &= \frac{\phi}{\square_w} \left| \frac{\psi}{\begin{pmatrix} w^\sharp \\ w\end{pmatrix}} \right. . \end{aligned}$$

- For any chart $\begin{pmatrix} g_\flat \\ g \end{pmatrix} : \begin{pmatrix} I \\ O \end{pmatrix} \rightrightarrows \begin{pmatrix} I' \\ O' \end{pmatrix}$, we need a square

$$\begin{array}{ccc} \mathbf{Sys}\begin{pmatrix} I \\ O \end{pmatrix} & \xrightarrow{\mathbf{Sys}\begin{pmatrix} g_\flat \\ g \end{pmatrix}} & \mathbf{Sys}\begin{pmatrix} I' \\ O' \end{pmatrix} \\ \Big\downarrow{\scriptstyle \mathrm{Behave}_{\mathsf{T}}\begin{pmatrix} I \\ O \end{pmatrix}} & \mathrm{Behave}_{\mathsf{T}}\begin{pmatrix} g_\flat \\ g \end{pmatrix} & \Big\downarrow{\scriptstyle \mathrm{Behave}_{\mathsf{T}}\begin{pmatrix} I' \\ O' \end{pmatrix}} \\ \mathbf{Vec}\left(\mathbf{Chart}\left(\begin{pmatrix} \mathsf{In}_{\mathsf{T}} \\ \mathsf{Out}_{\mathsf{T}} \end{pmatrix}, \begin{pmatrix} I \\ O \end{pmatrix}\right)\right) & \xrightarrow[\mathbf{Vec}\left(\mathbf{Chart}\left(\begin{pmatrix} \mathsf{In}_{\mathsf{T}} \\ \mathsf{Out}_{\mathsf{T}} \end{pmatrix}, \begin{pmatrix} g_\flat \\ g \end{pmatrix}\right)\right)}{} & \mathbf{Vec}\left(\mathbf{Chart}\left(\begin{pmatrix} \mathsf{In}_{\mathsf{T}} \\ \mathsf{Out}_{\mathsf{T}} \end{pmatrix}, \begin{pmatrix} I' \\ O' \end{pmatrix}\right)\right) \end{array}$$

  This will take any behavior from S to U with chart $\begin{pmatrix} g_\flat \\ g \end{pmatrix}$ and give the function which takes behaviors of shape T in S and gives the composite behavior of shape T in U. That is,

$$\mathrm{Behave}_{\mathsf{T}}\begin{pmatrix} g_\flat \\ g \end{pmatrix}(\mathsf{S}, \mathsf{U})(\psi) = \phi \mapsto \phi \mid \psi.$$

The naturality of this assignment follows from horizontal associativity in **Arena**.

Its a bit scary to see written out with all the names and symbols, but the idea is simple enough. We are composing two sorts of things: behaviors and systems. If we have some behaviors of shape T in our systems and their charts are compatible with a wiring pattern, then we get a behavior of the wired together system. If we have a chart, then behaviors with that chart give us a way of mapping forward behaviors of shape T.

The lax doubly indexed functor laws now tell us some facts about how these two sorts of composition interact.

- (Vertical Lax Functoriality) This asks us to suppose that we are wiring our systems together in two stages. The law then says that if we take a bunch of behaviors whose charts are compatible for the total wiring pattern and wire them together into a behavior of the whole system, this is the same behavior we get if we first noticed that they were compatible for the first wiring pattern, wired them together, then noticed that the result was compatible for the second wiring pattern, and wired that together. This means that nesting of wiring diagrams commutes with finding behaviors of our systems.
- (Horizontal Functoriality) This asks us to suppose that we have two charts and a behavior of each. The law then says that composing a behavior of shape T the composite of the behaviors is the same as composing it with the first one and then with the second one.
- (Functorial Interchange) This asks us to suppose that we have a pair of wiring patterns and compatible charts between them (a square in **Arena**). The law then says that if we take a bunch of behaviors whose charts are compatable according to

the first wiring pattern, wire them together, and then compose with a behavior of the second chart, we get the same thing as if we compose them all with behaviors of the first chart, noted that they were compatible with the second wiring pattern, and then wired them together.

Though it seems like it would be a mess of symbols to check these laws, they in fact fall right out of the laws for the double categories of arenas and matrices, and the functoriality of Proposition 3.70. That is, we've already built up all the tools we need to prove this fact, we just need to finish describing it.

- (Vertical Lax Functoriality) Suppose we have composable lenses $\binom{w^\sharp}{w} : \binom{I_1}{O_1} \leftrightarrows \binom{I_2}{O_2}$ and $\binom{u^\sharp}{u} : \binom{I_2}{O_2} \leftrightarrows \binom{I_3}{O_3}$. We need to show that

$$\mathrm{Behave}_{\mathsf{T}}^{\binom{w^\sharp}{w} \fatsemi \binom{u^\sharp}{u}} = \left(\mathrm{Behave}_{\mathsf{T}}^{\binom{u^\sharp}{u}} \mathbf{Sys}\left(\binom{w^\sharp}{w}\right)\right) \circ \left(\mathbf{VecChart}\left(\binom{\mathsf{In}_{\mathsf{T}}}{\mathsf{Out}_{\mathsf{T}}}, \binom{u^\sharp}{u}\right) \mathrm{Behave}_{\mathsf{T}}^{\binom{w^\sharp}{w}}\right).$$

  This follows immediately from vertical associativity in **Arena**, once both sides have been expanded out. Let S be a $\binom{I_1}{O_1}$-system, then

$$\begin{aligned}
\mathrm{Behave}_{\mathsf{T}}^{\binom{w^\sharp}{w} \fatsemi \binom{u^\sharp}{u}}(\mathsf{S})(\alpha, \phi) &= \mathrm{Behave}_{\mathsf{T}}^{\binom{w^\sharp}{w} \fatsemi \binom{u^\sharp}{u}}(\mathsf{S})\left(\frac{\beta}{\gamma}, \phi\right) && \text{by I} \\
&= \frac{\phi}{\frac{\beta}{\gamma}} \\
&= \frac{\frac{\phi}{\beta}}{\gamma} \\
&= \left(\mathrm{Behave}_{\mathsf{T}}^{\binom{u^\sharp}{u}} \mathbf{Sys}\left(\binom{w^\sharp}{w}\right)\right) \circ \left(\mathbf{Chart}\left(\binom{\mathsf{In}_{\mathsf{T}}}{\mathsf{Out}_{\mathsf{T}}}, \binom{u^\sharp}{u}\right) \mathrm{Behave}_{\mathsf{T}}^{\binom{w^\sharp}{w}}\right)(\gamma, \beta, \phi).
\end{aligned}$$

- (Horizontal Functoriality) This follows directly from horizontal associativity in **Arena**.
- (Functorial Interchange) This law will follow directly from interchange in the double category of arenas. Let α be a square in **Arena** of the following form:

$$\alpha = \begin{array}{ccc}
\binom{A^-}{A^+} & \xRightarrow{\binom{f_\flat}{f}} & \binom{B^-}{B^+} \\
\binom{j^\sharp}{j} \downarrow\uparrow & & \downarrow\uparrow \binom{k^\sharp}{k} \\
\binom{C^-}{C^+} & \xRightarrow[\binom{g^\sharp}{g}]{} & \binom{D^-}{D^+}
\end{array}$$

We need to show that

$$\mathrm{Behave}_{\mathsf{T}}^{\binom{j^\sharp}{j}} \left| \frac{\mathbf{Sys}(\alpha)}{\mathrm{Behave}_{\mathsf{T}}^{\binom{g_\flat}{g}}} = \frac{\mathrm{Behave}_{\mathsf{T}}^{\binom{f_\flat}{f}}}{\mathbf{VecChart}\left(\binom{\mathsf{In}_\mathsf{T}}{\mathsf{Out}_\mathsf{T}}, \alpha\right)} \right| \mathrm{Behave}_{\mathsf{T}}^{\binom{k^\sharp}{k}} \tag{3.73}$$

We can see both sides as natural transformations of the signature

$$\begin{array}{ccc}
\mathbf{Sys}\binom{A^-}{A^+} & \xrightarrow{\quad\mathbf{Sys}\binom{f_\flat}{f}\quad} & \mathbf{Sys}\binom{B^-}{B^+} \\
{\scriptstyle \mathrm{Behave}_{\mathsf{T}}^{\binom{A^-}{A^+}}}\downarrow & & \downarrow{\scriptstyle \mathbf{Sys}\binom{k^\sharp}{k}} \\
\mathbf{VecChart}\left(\binom{\mathsf{In}_\mathsf{T}}{\mathsf{Out}_\mathsf{T}}, \binom{A^-}{A^+}\right) & 3.73 & \mathbf{Sys}\binom{D^-}{D^+} \\
{\scriptstyle \mathbf{VecChart}\left(\binom{\mathsf{In}_\mathsf{T}}{\mathsf{Out}_\mathsf{T}}, \binom{j^\sharp}{j}\right)}\downarrow & & \downarrow{\scriptstyle \mathrm{Behave}_{\mathsf{T}}^{\binom{D^-}{D^+}}} \\
\mathbf{VecChart}\left(\binom{\mathsf{In}_\mathsf{T}}{\mathsf{Out}_\mathsf{T}}, \binom{C^-}{C^+}\right) & \xrightarrow[\mathbf{VecChart}\left(\binom{\mathsf{In}_\mathsf{T}}{\mathsf{Out}_\mathsf{T}}, \binom{g_\flat}{g}\right)]{} & \mathbf{VecChart}\left(\binom{\mathsf{In}_\mathsf{T}}{\mathsf{Out}_\mathsf{T}}, \binom{D^-}{D^+}\right)
\end{array}$$

Accordingly, let $\psi \in \mathbf{Sys}\binom{f_\flat}{f}(\mathsf{S}, \mathsf{U})$ be a behavior with chart $\binom{f_\flat}{f}$. We need to show that passing this through the left side of Eq. (3.73) equals the result of passing it through the right hand side. The result is an element of

$$\mathbf{VecChart}\left(\binom{\mathsf{In}_\mathsf{T}}{\mathsf{Out}_\mathsf{T}}, \binom{g_\flat}{g}\right)(\cdots, \cdots)$$

and is accordingly a function that takes in a pair of the following form:

$$(\square_j, \phi) = \left(\begin{array}{cc}
\binom{\mathsf{In}_\mathsf{T}}{\mathsf{Out}_\mathsf{T}} \overset{\binom{a_\flat}{a}}{\dashrightarrow} \binom{A^-}{A^+} & \binom{\mathsf{State}_\mathsf{T}}{\mathsf{State}_\mathsf{T}} \overset{\binom{\phi\circ\pi_2}{\phi}}{\dashrightarrow} \binom{\mathsf{State}_\mathsf{S}}{\mathsf{State}_\mathsf{S}} \\
\| \quad \downarrow\uparrow{\scriptstyle \binom{j^\sharp}{j}}, & \binom{\mathrm{update}_\mathsf{T}}{\mathrm{expose}_\mathsf{T}}\downarrow\uparrow \quad \downarrow\uparrow\binom{\mathrm{update}_\mathsf{S}}{\mathrm{expose}_\mathsf{S}} \\
\binom{\mathsf{In}_\mathsf{T}}{\mathsf{Out}_\mathsf{T}} \overset{\binom{c_\flat}{c}}{\Longrightarrow} \binom{C^-}{C^+} & \binom{\mathsf{In}_\mathsf{T}}{\mathsf{Out}_\mathsf{T}} \overset{\binom{a^\sharp}{a}}{\Longrightarrow} \binom{A^-}{A^+}
\end{array} \right)$$

The left hand side sends this pair to

$$\mathrm{Behave}_{\mathsf{T}}^{\binom{g_\flat}{g}}\left(\mathbf{Sys}(\alpha)(\psi)\right)\left(\mathrm{Behave}_{\mathsf{T}}^{\binom{j^\sharp}{j}}(\square_j, \phi)\right)$$

which equals, rather simply:

$$\frac{\phi}{\square_j} \left| \frac{\psi}{\alpha} \right. .$$

The right hand side sends the pair to

$$\mathrm{Behave}_{\mathsf{T}}^{\binom{k^\sharp}{k}}\left(\mathbf{VecChart}\left(\binom{\mathsf{In}_{\mathsf{T}}}{\mathsf{Out}_{\mathsf{T}}},\alpha\right)\left(\Box_j, \mathrm{Behave}_{\mathsf{T}}^{\binom{f_\flat}{f}}(\psi)(\phi)\right)\right)$$

which equals, rather simply:

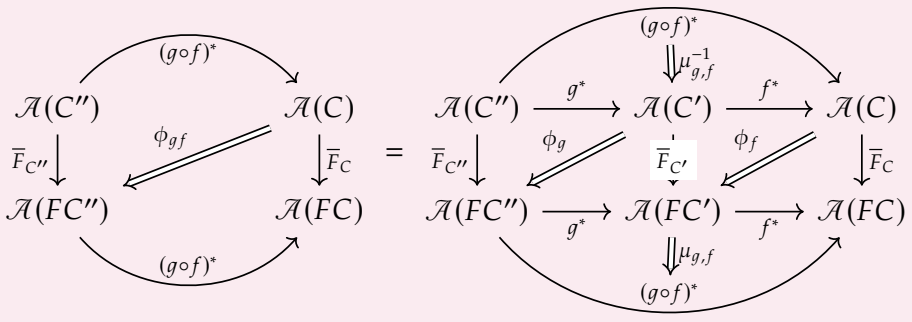
$$\frac{\phi \mid \psi}{\Box_j \mid \alpha}.$$

3.8 Change of doctine

Functoriality of the Grothendieck construction As with any categorical construction, the Grothendieck construction is functorial in its argument. To see how this works, we need to know what an indexed functor between indexed categories is.

Definition 3.74. Let $\mathcal{A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ and $\mathcal{B} : \mathcal{D}^{\mathrm{op}} \to \mathbf{Cat}$ be indexed categories. An indexed functor $(F, \overline{F}) : \mathcal{A} \to \mathcal{B}$ consists of a functor $F : \mathcal{C} \to \mathcal{D}$ together with a pseudo-natural transformation[a] $\overline{F} : \mathcal{A} \Rightarrow \mathcal{B} \circ F^{\mathrm{op}}$. Explicitly, this is:

- A functor $F : \mathcal{C} \to \mathcal{D}$.
- For each $C \in \mathcal{C}$, a functor $\overline{F}_C : \mathcal{A}(C) \to \mathcal{B}(FC)$.
- For each $f : C \to C'$ in $\mathcal{C}$, a natural isomorphism $\phi_f : \overline{F}_C \circ f^* \cong f^* \circ \overline{F}_{C'}$.
- (Psuedo-naturality) These naturality isomorphisms are required to satisfy a coherence condition: For $g : C' \to C''$, we need that



[a] As with the psuedo-functoriality of each indexed category, pseudo-naturality means naturality up to coherent isomorphism. In many of our cases, we will have bona-fide naturality.

Proposition 3.75 (Functoriality of the Grothendieck construction)**.** Let $(F, \overline{F}) : \mathcal{A} \to \mathcal{B}$ be an indexed double functor. Then there is a functor

$$\begin{pmatrix} \overline{F} \\ F \end{pmatrix} : \int^{C:\mathcal{C}} \mathcal{A}(C) \to \int^{D:\mathcal{D}} \mathcal{B}(D)$$

between their Grothendieck constructions given on objects by

$$\begin{pmatrix} \overline{F} \\ F \end{pmatrix} \begin{pmatrix} A \\ C \end{pmatrix} \coloneqq \begin{pmatrix} \overline{F}A \\ FC \end{pmatrix}.$$

Proof. □

Chapter 4

Polynomial functors and dynamics

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

–André Joyal, Summer 2020,
personal communication.

October 2, 2020

4.1 Introduction

In this chapter we will investigate a remarkable category called **Poly**. We will see its intimate relationship with both dynamic processes, data storage and transformations, and decision-making. But our story begins with something quite humble: middle school algebra.

$$y^2 + y + 1 \qquad \textit{polynomial} \tag{4.1}$$

All our polynomials will involve one variable, y, chosen for reasons we'll explain soon. Polynomials in one variable can be depicted as a set of mini-trees:

 *arena* (4.2)

More technically, mini-trees are called *corollas*, so what we're calling an *arena*, in this chapter, is a set of corollas.

Intuitively, one might think of each corolla as representing a *decision*. Associated to every decision is a set of *options*. The three decisions we exhibit in Eq. (4.2) are particularly interesting; they respectively have two options, one option, and no options. Having two options is familiar from life—it's the classic yes/no decision—as well

as from Claude Shannon's Information Theory. Having one option is also familiar theoretically and in life: "sorry, ya just gotta go through it." Having no options is when you actually don't get through it: it's "the end". While the corollas $1, y$, and y^2 are each interesting as decisions, their sum $y^2 + y + 1$ has very little theoretical interest; it's just a first example.

In a polynomial like $42y^3 + 17y$, the pure power term y^3 has a coefficient of 42 next to it, so the corolla with three leaves shows up 42 times[1] in the associated arena. Intuitively, there are 42 situations in which you have a decision with three options. When you put all those decisions together, you have the *arena* associated to p.

Here is a table of terminology. The first column is book-wide terminology, the second is algebraic terminology about polynomials as in (4.1), the third is about the pictures of corollas as in (4.2), and the fourth column is intuitive terminology.

Terminology

Arena	**Polynomial** p	**Decision Trees**	**Intuition**
Set of positions	$p(1)$	Set of roots	Set of situations
Interface in position i	Pure power summand $y^{p[i]}$	Corolla	Decision
Direction $c \in p[i]$	Element of $p[i]$	Leaf	Possibility

(4.3)

Exercise 4.4. Consider the polynomial $p := 2y^3 + 2y + 1$ and the associated arena.

1. Draw the arena whose name is p as a set of corollas.
2. How many roots does this arena have?
3. How many decisions does this represent?
4. For each corolla in the arena, say how many leaves it has.
5. For each decision, how many options does it have?

Now referring instead to the polynomial $q := y^{\mathbb{N}} + 4y$:

6. Does the polynomial q have a pure-power summand y^2?
7. Does the polynomial q have a pure-power summand y?
8. Does the polynomial q have a pure-power summand $4y$? ◊

Remark 4.5. Though a polynomial is a *functor* and an arena is a *dependent set*, they are so closely related that we do not make a distinction between a polynomial p and its arena; they are two different syntaxes for the same object.

Exercise 4.6. If you were a suitor choosing the arena you love, aesthetically speaking, which would strike your interest? Answer by circling the associated polynomial:

1. $y^2 + y + 1$
2. $y^2 + 3y^2 + 3y + 1$

[1] In standard font, 42 represents the usual natural number. In sans serif font $\mathsf{42}$ represents the set $\mathsf{42} = \{1, 2, \ldots, 42\}$ with 42 elements.

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

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

Before we can really get into this story, let's summarize where we're going: polynomials are going to have really surprising applications to dynamics, data, and decision. Remember, we spoke superlatively of **Poly** at the beginning of this chapter,

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

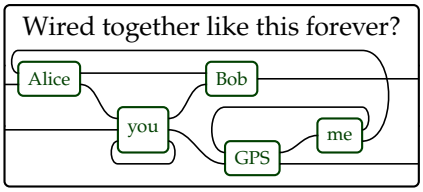
and we have not yet begun to deliver. So let's introduce some of the applications and mathematics to come.

4.1.1 Dynamical systems

Throughout this book, we've seen dynamical systems—machines of various sorts—which have an internal state that can be read out to other systems, as well as updated based on input received from other systems. In the context of this chapter, we'll be looking mainly at deterministic systems, but with a lot of interesting new options:

1. The interface of the system can change shape through time.
2. The wiring diagram connecting a bunch of systems can change in time.
3. One can speed up the dynamics of a system.
4. One can introduce "effects", i.e. as defined by monads on **Set**.
5. The dynamical systems on any interface form a topos.

To give some intuition for the first two, imagine yourself as a system, wired up to other systems. You have some input ports: your eyes, your ears, etc., and you have some output ports: your speech, your gestures, etc. And you connect with other systems: your family, your colleagues, the GPS of your phone, etc.



(4.7)

We wrote a little question for you at the top of the diagram. Isn't there something a little funny about wiring diagrams? Maybe for old-fashioned machines, you would wire things together once and they'd stay like that for the life of the machine. But my phone connects to different wifi stations at different times, I drop my connection to Alice for weeks at a time, etc. So wiring diagrams should be able to change in time; **Poly** will let us do that.

Example 4.8. Here are some familiar circumstances where we see wiring diagrams changing in 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 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, i.e. 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 connection 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 4.152 to 4.154.

Exercise 4.9. Think of another example where systems are sending each other information, but where the sort of information or who it's 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 wiring patterns in your situation. ◊

But there's more that's intuitively wrong or limiting about the picture in (4.7). 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 phone can't receive GPS. Sometimes I extend my hand to give or receive an object from another person, but sometimes I don't. My ports themselves change in time. Sometimes I even use my output ports to determine the wiring pattern of other machines. We will be able to say all this using **Poly**.

And there's even more that's wrong with the above description. Namely, when I move my eyes, that's actually something you can see—e.g. whether I'm looking at you. When I turn around, I see different things, and *you can notice I'm turned around*! When I use my muscles or mouth to express things, my very position changes: my tongue moves, my body moves. So my output is actually achieved by changing position. The model in **Poly** will be able to express this too.

Example 4.10. 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 intuitively consists of two situations, one with n-many options, and the other with only one option.

The point is, however, is that the other eyeballs can tell if e is opened 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!

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

But **Poly** is very versatile in its applications. Next we'll show how it relates to information storage and translation.

4.1.2 Data

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

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

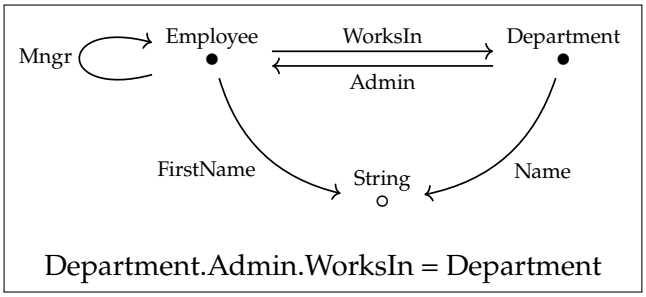
Employee	FirstName	WorksIn	Mngr
1	Alan	101	2
2	Ruth	101	2
3	Carla	102	3

Department	Name	Admin
101	Sales	1
102	IT	3

(4.11)

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

$\mathcal{C} :=$



(4.12)

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

What's called $\mathcal{C}$ in (4.12) is a *finitely presented category*, and the database instance presented in (4.11) correspond to a functor $I\colon \mathcal{C} \to \mathbf{Set}$; it sends the object Employee $\in \mathcal{C}$ to the set $I(\text{Employee}) := \{1, 2, 3\}$. It sends the morphism Mngr: Employee $\to$ Employee to the function $I(\text{Mngr})\colon \{1, 2, 3\} \to \{1, 2, 3\}$ given by $I(1) = 2$, $I(2) = 2$, and $I(3) = 3$. A functor $\mathcal{C} \to \mathbf{Set}$ is called a *copresheaf on* $\mathcal{C}$. So the story of database schemas and their data can be based on the story of categories and their copresheaves.

There's a very important thing that we do with databases: we query them. We ask them questions like "tell me everyone that's either the admin person of the Math department or their manager".

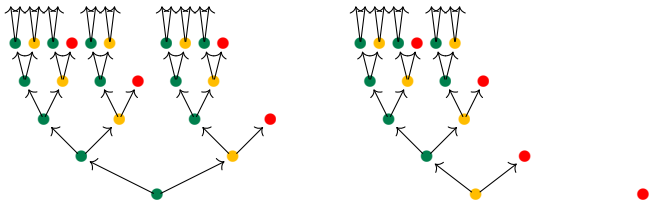
```
FOR    d: Department, e: Employee
WHERE  Name(d)="Math" AND
       (e=Admin(d) OR e=Mngr(Admin(d)))
RETURN FirstName(e)
```

This sort of question is formally called a "union of conjunctive queries". We will see this sort of query is intimately connected with **Poly**.

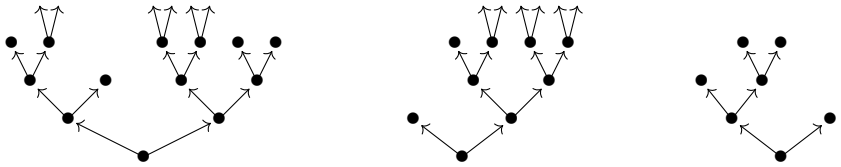
We will also see how databases can be conceived in terms of dynamical systems.

4.1.3 Decisions

Consider the following three trees, the first two are infinite (though that's hard to draw):



These are patterned examples—and we'll understand what this pattern is more clearly in **??**—of what we will call *decision streams*. Decision streams form the objects in a category that also includes the following level-3 abbreviations of binary decision streams (the third of which is a finite stream):

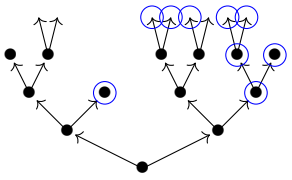


The set of such decision streams in fact form the objects of a category with very nice properties (it's a topos), which we call $\mathbf{Sys}(y^2+1)$. The idea is that every corolla in these diagrams has either two options, corresponding to y^2, or no options, corresponding to $1 = y^0$.

Exercise 4.13.

1. Draw a level-3 abbreviation of a decision stream of type $y^2 + y^0$.
2. Draw a level-4 abbreviation of a decision stream of type y.
3. Draw a level-3 abbreviation of a decision stream of type $\mathbb{N}y^2$ by labeling every node with a natural number. ◊

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



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

4.1.4 Implementation

Everything we talk about can actually be implemented in a computer without much difficulty, at least if you have access to a language that supports dependent types. We will continue to use Agda.

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

Luckily, the agda code is pretty easy to understand.

```
record Arena : Set where  -- a arena consists of
  field                   -- two fields
    decn : Set            -- one called decn, a set, and
    posy : pos -> Set     -- one called posy,
                          -- a set that depends on decn
```

4.1.5 Mathematical theory

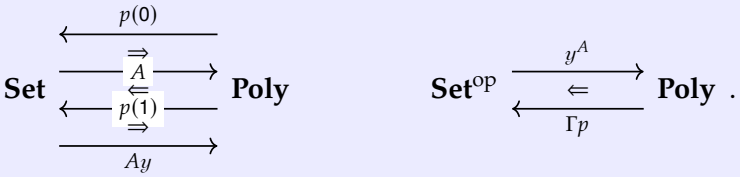
The applications of **Poly** are quite diverse and interesting, including dynamics, data, and decisions. However it is how the mathematics of **Poly** supports these applications that is so fantastic. For experts, here are some reasons for the excitement.

Proposition 4.14. **Poly** is a biCartesian closed category, meaning it has sums, products, and exponential objects. It supports the simply typed lambda calculus.

In fact **Poly** has infinite coproducts, infinite products, and is completely distributive. We'll explain all of this in **??**; for now we continue with the highlights.

Proposition 4.15. 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. Moreover $\otimes$ is also a closed monoidal structure and it distributes over $+$.

Proposition 4.16. **Poly** has an adjoint quadruple with **Set** and an adjoint pair with $\mathbf{Set}^{\mathrm{op}}$:



Each functor is labeled by where it sends $p \in \mathbf{Poly}$ or $A \in \mathbf{Set}$.

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

Proposition 4.17. The functor $\mathbf{Poly} \to \mathbf{Set}$ given by $p \mapsto p(1)$ is a monoidal fibration.

In fact it's a monoidal $*$-bifibration in the sense of [**shulman2008framed**]. But here's where things get really interesting.

Proposition 4.18 (Ahman-Uustalu)**.** Comonoids in $(\mathbf{Poly}, y, \circ)$ are categories (up to isomorphism).

Proposition 4.19 (Niu)**.** The category $\mathbf{Comon}(\mathbf{Poly})$ has finite coproducts and products, and coproducts in $\mathbf{Comon}(\mathbf{Poly})$ agree with those in **Cat**.

Proposition 4.20. Suppose that the category $\mathcal{C}$ corresponds under Proposition 4.18, to the comonoid $\mathscr{C}$. Then there is an equivalence of categories

$$\mathbf{Bimod}(\mathscr{C}, 0) \cong \mathcal{C}\text{-}\mathbf{Set}$$

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

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

Proposition 4.21 (Garner)**.** For any categories $\mathcal{C}$ and $\mathcal{D}$, there is an equivalence of categories

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

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

Proposition 4.22 (Niu)**.** For any category $\mathcal{C}$ the category of left $\mathscr{C}$-modules is equivalent to the category of functors $\mathcal{C} \to \mathbf{Poly}$,

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

Proposition 4.23. The functor **Comon**(**Poly**) → **Poly** has a right adjoint

$$\mathbf{Poly} \underset{u_-}{\overset{\mathscr{K}_-}{\underset{\longleftarrow}{\overset{\longrightarrow}{\Leftarrow}}}} \mathbf{Comon}(\mathbf{Poly}) \ ,$$

called the *cofree comonoid* construction. It is lax monoidal with respect to ⊗.

Proposition 4.24. The category **Comon**(**Poly**) has a third symmetric monoidal structure $(y, \otimes)$, and the functor $u_-\colon (\mathbf{Comon}(\mathbf{Poly}), y, \otimes) \to (\mathbf{Poly}, y, \otimes)$ is strong monoidal.

Proposition 4.25 (Niu)**.** For any polynomial p, the category

$$\mathbf{Sys}(p) \cong \mathscr{K}_p\text{-}\mathbf{Set}$$

of dynamical systems on p forms a topos.

Proposition 4.26. A morphism $p \to q$ of polynomials induces a pre-geometric morphism between their respective toposes

$$\mathbf{Sys}(p) \underset{\longleftarrow}{\overset{\longrightarrow}{\Rightarrow}} \mathbf{Sys}(q) \ .$$

If you skipped over any of that—or all of that—it'll be no problem whatsoever! We will get to everything in the current section, 4.1.5, over the next few chapters. To briefly summarize the results we have just seen, they say in particular that there is a full-fledged logic of dynamical systems inhabiting any interface (Proposition 4.25), that these logics can be combined and compared (Proposition 4.26), and that the whole story of dynamics carries a strong connection to database theory. There are many avenues for study, but we need to push forward.

Let's begin.

4.2 Introduction to Poly as a category

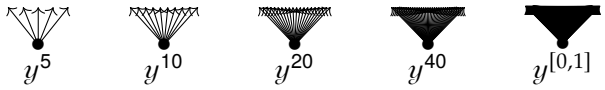
In this section, we will set down the basic category-theoretic story of **Poly**, so that we can have a firm foundation from which to speak about dynamical systems, decisions, and data. We begin with the category **Set** of sets and functions, and what is arguably the fundamental theorem of category theory, the Yoneda lemma.

4.2.1 Representable functors and the Yoneda lemma

Definition 4.27. Given any set S, we denote by $y^S \colon \mathbf{Set} \to \mathbf{Set}$ the functor that sends any set X to the set $X^S = \mathbf{Set}(S, X)$, and sends any function $h \colon X \to Y$ to the function $h^S \colon X^S \to Y^S$, the one that sends $g \colon S \to X$ to $g \fatsemi h \colon S \to Y$.

We refer to y^S as the functor *represented by* S.

The symbol y stands for Yoneda, for reasons we will get to in Lemma 4.36. For now, here are some pictures for your eyes to gaze at; they are the poly's corresponding to various representables:



(4.28)

Example 4.29. The functor that sends every set X to $X \times X$, and sends $h \colon X \to Y$ to $(h \times h) \colon (X \times X) \to (Y \times Y)$ is representable.

Exercise 4.30. For each of the following functors $\mathbf{Set} \to \mathbf{Set}$, say if it is representable or not; if it is, say what the representing set is.

1. The identity functor $X \mapsto X$.
2. The constant functor $X \mapsto 2$.
3. The constant functor $X \mapsto 1$.
4. The constant functor $X \mapsto 0$.
5. The functor $X \mapsto X^{\mathbb{N}}$.
6. The functor $X \mapsto 2^X$. ◊

Proposition 4.31. For any function $f \colon R \to S$, there is an induced natural transformation $y^f \colon y^S \to y^R$; on any set X the X-component $X^f \colon X^S \to X^R$ is given by sending $g \colon S \to X$ to $f \fatsemi g \colon R \to X$.

Exercise 4.32. Prove that for any function $f \colon R \to S$, what we said was a natural transformation in Proposition 4.31 really is natural. That is, for any function $h \colon X \to Y$, show that the following diagram commutes:

$$\begin{array}{ccc} X^S & \xrightarrow{h^S} & Y^S \\ {\scriptstyle X^f}\downarrow & ? & \downarrow{\scriptstyle Y^f} \\ X^R & \xrightarrow[h^R]{} & Y^R \end{array}$$

◊

Exercise 4.33. Let $X = \{a, b, c\}$. For each of the following sets R, S and functions $f : R \to S$, describe the X-component of, i.e. the function $X^S \to X^R$ coming from, the natural transformation $y^f : y^S \to y^R$.

1. $R = 5$, $S = 5$, $f = \mathrm{id}$. (Here you're supposed to give a function called $X^{\mathrm{id}_5} : X^5 \to X^5$.)
2. $R = 2$, $S = 1$, f is the unique function.
3. $R = 1$, $S = 2$, $f(1) = 1$.
4. $R = 1$, $S = 2$, $f(1) = 2$.
5. $R = 0$, $S = 5$, f is the unique function.
6. $R = \mathbb{N}$, $S = \mathbb{N}$, $f(n) = n + 1$. ◊

Exercise 4.34. Show that the construction in Proposition 4.31 is functorial

$$y^- : \mathbf{Set}^{\mathrm{op}} \to \mathbf{Set}^{\mathbf{Set}}, \tag{4.35}$$

as follows.

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

Lemma 4.36 (Yoneda lemma)**.** Given a functor $F : \mathbf{Set} \to \mathbf{Set}$ and a set S, there is an isomorphism

$$F(S) \cong \mathsf{Nat}(y^S, F) \tag{4.37}$$

where Nat denotes the set of natural transformations. Moreover, (4.37) is natural in both S and F.

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

For any element $a \in F(S)$, there is a natural transformation $m_a : y^S \to F$ whose component on X is the function $X^S \to F(X)$ given by sending $g : S \to X$ to $F(g)(a)$. In Exercise 4.38 we ask you to show that this is natural in X and that these two constructions are mutually inverse. □

Exercise 4.38. Whoever solves this exercise can say they've proved the Yoneda Lemma.

1. Show that for any $a \in F(S)$, the maps $X^S \to F(X)$ given as in Lemma 4.36 are natural in X.
2. Show that the two mappings from Lemma 4.36 are mutually inverse.
3. As a corollary of Lemma 4.36, show that $y^- : \mathbf{Set}^{\mathrm{op}} \to \mathbf{Set}^{\mathbf{Set}}$ is fully faithful, in particular that there is an isomorphism $\mathsf{Nat}(y^S, y^T) \cong S^T$. ◊

4.2.2 Polynomials: sums of representables

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

So here's the deal. All of our polynomials will be polynomials in one variable, y; every other letter or number that shows up will represent a set.[2] For example, in the following crazy polynomial

$$p \coloneqq \mathbb{R}y^{\mathbb{Z}} + 3y^3 + Ay + \sum_{i \in I} Q_i y^{R_i + Q_i^2}, \tag{4.39}$$

$\mathbb{R}$ denotes the set of real numbers, $\mathbb{Z}$ denotes the set of integers, 3 denotes the set $\{1, 2, 3\}$, and A, I, Q_i, and R_i denote some arbitrary sets that should have already been defined in order for (4.39) to make sense.

The polynomials we already understand at this point are the pure-power polynomials y^A for some set A; they are representable functors $y^A \colon$ **Set** $\to$ **Set**. So to understand general polynomials like p, we just need to understand sums of functors. It will be useful to discuss products of functors at the same time. However, we haven't discussed sums and products of sets yet, so we need to do that first.

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

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

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

Definition 4.40 (Dependent sums and products of sets)**.** Let I be a set and $X \colon I \to$ **Set** be an I-indexed collection of sets. The sum $\sum_{i \in I} X_i$ (respectively, the *product* $\prod_{i \in I} X_i$)

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

of this collection is the set

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

Example 4.41. If $I = 2 = \{1,2\}$ then a collection $X\colon I \to \mathbf{Set}$ is just two sets, say $X_1 = \{a,b,c\}$ and $X_2 = \{c,d\}$. The 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 number of elements (cardinality) will always be the sum of the numbers of elements in X_1 and X_2. Similarly,

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

Exercise 4.42. 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 $I := \varnothing$ and $X\colon \varnothing \to \mathbf{Set}$ is the unique empty collection of sets.

1. Is it true that $X_i = 1$ for each $i \in I$?
2. Show that there is an isomorphism of sets $0 \cong \sum_{i\in\varnothing} X_i$.
3. Show that there is an isomorphism of sets $1 \cong \prod_{i\in\varnothing} X_i$. ◊

Exercise 4.43. Let $X_{(-)} : I \to \mathbf{Set}$ be a set depending on an $i \in I$. There is a projection function $\pi_1 : \sum_{i\in I} X_i \to I$ defined by $\pi_1(i,x) = i$.

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

$$\prod_{i\in I} X_i \cong \{s : I \to \sum_{i\in I} X_i \mid \pi_1 \circ s = \mathrm{id}_I\}.$$

◊

Where are we, and where are we going? We've define dependent sums and products of sets; that's where we are. 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 define sums of functors $\mathbf{Set} \to \mathbf{Set}$. But we might as well define products of functors at the same time, because they'll very much come in handy.

Dependent sums and products of functors Set → Set.

Definition 4.44 (Dependent sums and products of functors **Set** → **Set**)**.** For any two functors $F, G\colon \mathbf{Set} \to \mathbf{Set}$, let

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

denote the functors that respectively assign to each $X \in \mathbf{Set}$ the sets

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

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

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

denote the functors that respectively assign to each $X \in \mathbf{Set}$ the sum and product of sets

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

We denote by $0, 1\colon \mathbf{Set} \to \mathbf{Set}$ the constant functors, which respectively assign 0 and 1 to each $X \in \mathbf{Set}$.

Proposition 4.45. Referring to the notation in Definition 4.44, the functors 0 and 1 are respectively an initial object and a terminal object in $\mathbf{Set}^{\mathbf{Set}}$, the operations $+$ and $\times$ are respectively a binary coproduct and a binary product in $\mathbf{Set}^{\mathbf{Set}}$, and the operations $\sum_{i\in I}$ and $\prod_{i\in I}$ are arbitrary coproducts and products in $\mathbf{Set}^{\mathbf{Set}}$.

Proof. By Example 4.41 and Exercise 4.42, it suffices to show that $\sum_{i\in I} F_i$ and $\prod_{i\in I} F_i$ are a sum and product in $\mathbf{Set}^{\mathbf{Set}}$. This itself is a special case of a more general fact, where sums and products are replaced by arbitrary colimits and limits, and where $\mathbf{Set}^{\mathbf{Set}}$ is replaced by an arbitrary functor category $\mathcal{C}^{\mathcal{D}}$, where $\mathcal{C}$ is a category that (like **Set**) has limits and colimits; see [**macLane1992sheaves**]. □

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

Poly, the category of polynomial functors

Definition 4.46 (Polynomial functors)**.** A *polynomial functor* is a functor $p\colon \mathbf{Set} \to \mathbf{Set}$ such that there exists a set I, sets $(p[i])_{i\in I}$, and an isomorphism

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

to a sum of representables. A *morphism of polynomial functors* is simply a natural transformation $p \to q$.

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

Example 4.47. Consider the polynomial $y^2 + 2y + 1$. It denotes a functor **Set** $\to$ **Set**; what does this functor do to the set $X := \{a, b\}$? To be very precise and pedantic, let's say

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

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

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

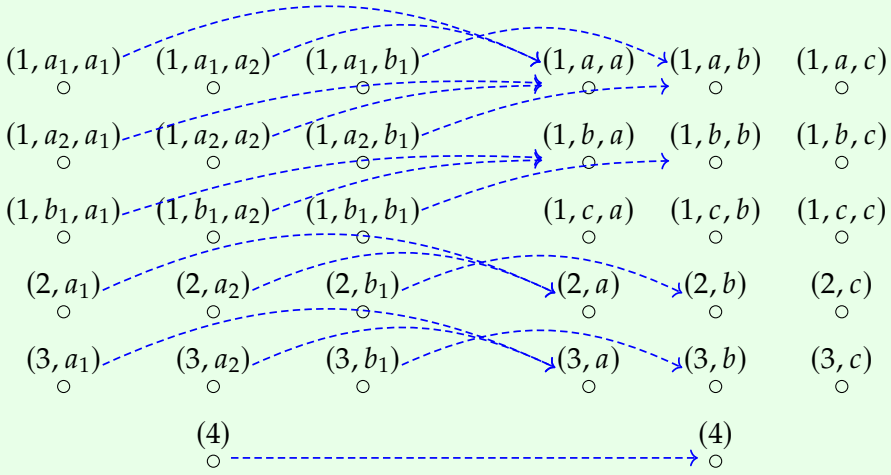
It has $(2^2 + 2 + 2 + 1)$-many, i.e. 9, elements. The representable summand y^A throws in all A-tuples from X, but it's indexed by the name of the summand. In particular if $A = \varnothing$ then it just records the empty tuple at that summand.

Exercise 4.48. In the pedantic style of Example 4.47, 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 $X := \mathbb{N}$. ◊

Example 4.49. Suppose $p := y^2 + 2y + 1$. As a functor **Set** $\to$ **Set** it should be able to act not only on sets but on functions. Let $X := \{a_1, a_2, b_1\}$, $Y := \{a, b, c\}$ and $f : X \to Y$ the function sending $a_1, a_2 \mapsto a$ and $b_1 \mapsto b$. The induced function $p(f) : p(X) \to p(Y)$ is

shown below



Exercise 4.50. Let $p = y^2 + y$. Choose a function $f : 1 \to 2$ and write out the induced function $p(f) : p(1) \to p(2)$. ◊

Proposition 4.51. Let $p \coloneqq \sum_{i \in I} y^{p[i]}$ be an arbitrary polynomial functor. Then $I \cong p(1)$, so there is an isomorphism of functors

$$p \cong \sum_{i \in p(1)} y^{p[i]}. \tag{4.52}$$

Proof. We need to show that $I \cong p(1)$; the latter claim follows directly. In Exercise 4.42 it was shown that $I \cong \sum_{i \in I} 1$, so we just need to show that $(y^A)(1) \cong 1$ for any $A \in \mathbf{Set}$. But $1^A \cong 1$ because there is a unique function $A \to 1$ for any A. □

The notation in (4.52) will be how we denote arbitrary polynomials from now on.

Exercise 4.53. We saw in Proposition 4.51 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? ◊

4.2.3 Morphisms between polynomial functors are dependent lenses

For a seasoned category theorist, we've already said enough to calculate the set $\mathbf{Poly}(p, q)$ of morphisms between polynomials p and q: it's the set of natural transformations between p and q, which are sums of representable functors. But it is worth going over this subject at a more leisurely pace, because it helps us insert some intuition that will be very useful later.

In particular, we're going to think about polynomials like p and q using the intuition from the beginning of the chapter, around the table in (4.3). We said that we'd think of p as a arena: a set of decisions you can make and what options you have. That is, $p(1)$ is conceived as a set of decisions, and for every decision $i \in p(1)$, we conceive of the set $p[i]$ as the options or *options* available in that decision. Here's how we write p as an arena:

$$p(1) \xrightarrow{p[-]} \mathbf{Set}$$

Again, every decision $i \in p(1)$ gets a set $p[i]$ of options: that's the whole content of p.

Definition 4.54. An *arena* for the dependent doctrine is a *dependent set* $A^-_{(-)} : A^+ \to \mathbf{Set}$; that is, functor from the discrete category on a set A^+ into the category of sets. the set A^-_a that each direction a^- lives in depends on the position $a \in A^+$. We will write the arena $A^- : A^+ \to \mathbf{Set}$ as

$$\begin{pmatrix} A^-_a \\ a \in A^+ \end{pmatrix}.$$

We have just noted that every polynomial p gives an arena $\left(\begin{smallmatrix} p[i] \\ i \in p(1) \end{smallmatrix}\right)$ whose positions are the decisions in the polynomial and whose directions are the options.

The maps $f : p \to q$ of polynomials are then the ways to *delegate* p's decisions to q. Every one of p's decisions, say $i \in p(1)$, is passed forward to a decision $j \in q(1)$ for q to make, and every choice $d \in q[j]$ that q could make among its options is passed back as some choice $c \in p[i]$ among p's options.

```
record Delegation (w : Arena) (w' : Arena) : Set where
  field
    passfwd : (decn p) -> (decn q)
    passbck : (d : decn p) -> posy q (passfwd d) -> posy p d
```

This structure of passing information forward and backwards should look familiar; its a sort of lens! The difference between this sort of lens and the sort that we saw in earlier chapters is that the *signature* of the passback can depend on which decision one is using. We will therefore refer to these lenses as *dependent lenses*.

Definition 4.55. A *dependent lens* $\left(\begin{smallmatrix} f^\sharp \\ f \end{smallmatrix}\right) : \left(\begin{smallmatrix} A^-_a \\ a \in A^+ \end{smallmatrix}\right) \leftrightarrows \left(\begin{smallmatrix} B^-_b \\ b \in B^+ \end{smallmatrix}\right)$ consists of

- a *passforward* function $f^+ : A^+ \to B^+$, and
- a dependent *passback* fucntion $f^\sharp : (a \in A^+) \to (B^-_{f(a)} \to A^-_a)$. That is, for every $a \in A^+$, we have a passback function $f^\sharp(a, -) : B^-_{f(a)} \to A^-_a$.

The composite of the dependent lens $\left(\begin{smallmatrix} f^\sharp \\ f \end{smallmatrix}\right) : \left(\begin{smallmatrix} A^-_a \\ a \in A^+ \end{smallmatrix}\right) \leftrightarrows \left(\begin{smallmatrix} B^-_b \\ b \in B^+ \end{smallmatrix}\right)$ with the dependent

lens $\begin{pmatrix} f^\sharp \\ f \end{pmatrix} : \begin{pmatrix} B_b^- \\ b \in B^+ \end{pmatrix} \leftrightarrows \begin{pmatrix} C_c^- \\ c \in C^+ \end{pmatrix}$ is given by

$$\begin{pmatrix} g^\sharp \\ g \end{pmatrix} \circ \begin{pmatrix} f^\sharp \\ f \end{pmatrix} := \begin{pmatrix} (a, c^-) \mapsto f^\sharp(a, g^\sharp(f(a), c^-)) \\ g \circ f \end{pmatrix}.$$

This gives us the category of *dependent lenses* **DLens**.

Proposition 4.56. Consider the functor $\mathbf{Set}^{\mathrm{op}} \to \mathbf{Cat}$ sending each set A to the slice category $\mathbf{Set}/A$, and sending each $f : B \to A$ to the functor $f^* : \mathbf{Set}/A \to \mathbf{Set}/B$. Then **DLens** is equivalent to the category *LSpiz: notation?* of lenses for this doctrine.

Proof. We define a functor $F : \mathbf{DLens} \to L$ as follows. An arena $A := \begin{pmatrix} A_a^- \\ a \in A^+ \end{pmatrix} \in \mathrm{Ob}\,\mathbf{DLens}$, assigns a set A_a^- to each $a \in A^+$; let $X := \sum_{a \in A^+} A_a^-$ with its obvious map $X \to A^+$. Then $F(A) := \begin{pmatrix} X \\ A \end{pmatrix}$ is notation for an object in L. We leave it to the reader to give the functor on morphisms, and show that it is fully faithful and essentially surjective. □

Exercise 4.57. Complete the proof of Proposition 4.56 as follows.

1. To what morphism in L does F assign the morphism $\begin{pmatrix} f^\sharp \\ f \end{pmatrix} : \begin{pmatrix} A_a^- \\ a \in A^+ \end{pmatrix} \leftrightarrows \begin{pmatrix} B_b^- \\ b \in B^+ \end{pmatrix}$?
2. Show that F is fully faithful.
3. Show that F is essentially surjective. ◊

Remark 4.58. Let $p := \sum_{i \in p(1)} y^{p[i]}$ and $q := \sum_{j \in q(1)} y^{q[j]}$ be polynomials. We can think of a dependent lens $p \to q$ as in Definition 4.55 diagrammatically as follows:

$$\begin{array}{ccc} p(1) & \xrightarrow{f_1} & q(1) \\ & \overset{f^\sharp}{\Leftarrow} & \\ p[-] \searrow & & \swarrow q[-] \\ & \mathbf{Set} & \end{array} \tag{4.59}$$

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

Dependent lenses are indeed a generalization of the lenses we saw in Definition 1.46.

Proposition 4.60. There is a fully faithful inclusion $\mathbf{Lens}_{\mathbf{Set}} \hookrightarrow \mathbf{DLens}$ interpreting the coKleisli arena $\begin{pmatrix} A^- \\ A^+ \end{pmatrix}$ as the *constant* dependent arena $\begin{pmatrix} A^- \\ a \in A^+ \end{pmatrix}$ (that is, the constant functor $a \mapsto A^- : A^+ \to \mathbf{Set}$).

Proof. We just need to check that the definition of dependent lens specializes appropriately to the definition of lens. So, suppose we have constant dependent arenas $\begin{pmatrix} A^- \\ a \in A^+ \end{pmatrix}$ and $\begin{pmatrix} B^- \\ b \in B^+ \end{pmatrix}$. Then a dependent lens $\begin{pmatrix} f^\sharp \\ f \end{pmatrix}$: □

Lemma 4.61. Using $\sum$s and $\prod$s, we can give a slick definition of dependent lens. We have a bijection

$$\mathbf{DLens}\left(\begin{pmatrix} A_a^- \\ a \in A^+ \end{pmatrix}, \begin{pmatrix} B_b^- \\ b \in B^+ \end{pmatrix}\right) \cong \prod_{a \in A^+} \sum_{b \in B^+} (A_a^-)^{(B_b^-)}$$

Proof. This is a special case of the upcoming Proposition 4.74, so we'll just sketch the proof here.

A dependent lens $\begin{pmatrix} f^\sharp \\ f \end{pmatrix} : \begin{pmatrix} A_a^- \\ a \in A^+ \end{pmatrix} \leftrightarrows \begin{pmatrix} B_b^- \\ b \in B^+ \end{pmatrix}$ consists of a map $f : A^+ \to B^+$ and a depedent map $f^\sharp : (a \in A^+) \to (B^-_{f(a)} \to A_a^-)$, while an element of the right ahnd side is dependent function

$$(f, f^\sharp) : (a \in A^+) \to \sum_{b \in B^+} (B_b^- \to A_a^-)$$

which takes an element of a and yields a pair $(f(a), f^\sharp(a, -))$. These are just two different ways of pairing up the data. □

It seems we are observing that the category of polynomial functors is equivalent to the category of dependent lenses! Let's state this mathematically and prove it.

Proposition 4.62. Taking a polynomial p to its arena $\begin{pmatrix} p[i] \\ i \in p(1) \end{pmatrix}$ gives an equivalence between **Poly** and the category of dependent lenses.

To summarize, we have an isomorphism

$$\mathbf{Poly}(p, q) \cong \prod_{i \in p(1)} \sum_{j \in q(1)} p[i]^{q[j]} \cong \mathbf{DLens}\left(\begin{pmatrix} p[i] \\ i \in p(1) \end{pmatrix}, \begin{pmatrix} q[j] \\ j \in q(1) \end{pmatrix}\right) \tag{4.63}$$

and composition of polynomial morphisms corresponds to composition of dependent lenses.

Proof. We have an assignment $p \mapsto \begin{pmatrix} p[i] \\ i \in p(1) \end{pmatrix}$ which sends an object of **Poly** — a polynomial functor — to an object of the category of dependent lenses — an arena. We want to show that this object assignment gives an equivalence of categories. First, we can note that it is surjective up to isomorphism on objects, since to every arena $\begin{pmatrix} A_a^- \\ a \in A^+ \end{pmatrix}$ we get the polynomial functor

$$\sum_{a \in A^+} y^{A_a^-}$$

and if we take the arena corresponding to the polynomial, we back where we started since $\sum_{a \in A^+} 1^{A_a^-} \cong A^+$.

Next, we'll show that the set of natural transformations $\mathbf{Poly}(p, q)$ between polynomial functors p and q is isomorphic to the set of dependent lenses between their arenas.

We have the following isomorphisms

$$\begin{aligned}
\mathbf{Poly}(p,q) &= \mathsf{Nat}\Big(\sum_{i\in p(1)} y^{p[i]}, \sum_{j\in q(1)} y^{q[j]}\Big) && \text{Definitions}\\
&\cong \prod_{i\in p(1)} \mathsf{Nat}\Big(y^{p[i]}, \sum_{j\in q(1)} y^{q[j]}\Big) && \text{Univ. prop. of coproduct}\\
&\cong \prod_{i\in p(1)} \sum_{j\in q(1)} p[i]^{q[j]} && \text{Yoneda lemma}
\end{aligned}$$

This proves the first part of we called the summary. It says that a morphism $p \to q$ consists of: for every $i \in p(1)$ both a choice of $j \in q(1)$ and a function from $q[j] \to p[i]$; the first part determines f (i.e. $f(i) = j$) and the second part just gives $f_i^\sharp$ directly for each i.

We can now note that by Lemma 4.61, $\prod_{i\in p(1)} \sum_{j\in q(1)} p[i]^{q[j]}$ is isomorphic to the set of dependent lenses $\left(\begin{smallmatrix} p[i] \\ i\in p(1)\end{smallmatrix}\right) \leftrightarrows \left(\begin{smallmatrix} q[j] \\ j\in q(1)\end{smallmatrix}\right)$.

Finally, it remains to show that these isomorphisms are functorial. We leave this to Exercise 4.64 □

Exercise 4.64. Complete the proof of Proposition 4.62. ◊

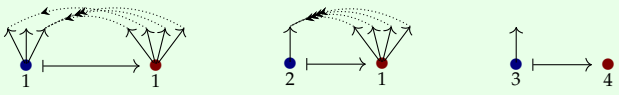
Remark 4.65. The fact in (4.63) that a natural transformation between polynomial functors is equivalently a dependent lens will be quite important to us; we will call it our *main formula* for the set of polynomial maps.

In fact, this formula will be so important to us that we will often blur the difference between a dependent lens $\left(\begin{smallmatrix} f^\sharp \\ f\end{smallmatrix}\right) : \left(\begin{smallmatrix} p[i] \\ i\in p(1)\end{smallmatrix}\right) \leftrightarrows \left(\begin{smallmatrix} q[j] \\ j\in q(1)\end{smallmatrix}\right)$ with the natural transformation $p \to q$ it corresponds to.

Example 4.66. Let $p := y^3 + 2y$ and $q := y^4 + y^2 + 2$

To give a dependent lens $p \to q$, one delegates each p-decision $i \in p(1)$ to a q-decision $j \in q(1)$, and then each option in $q[j]$ is passed back to one in $p[i]$.

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



This represents one morphism $p \to q$.

So how many different morphisms are there from p to q? One way to figure it out is to use the formula (4.63), but that might need some explaining; we go through such notation in an interlude after Exercise 4.67. For now, we can just count "pre-theoretically".

The first p-decision can be delegated to q-decision 1, 2, 3, or 4. Delegating it to the first q-decision 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 all the ways to delegate the first p-decision: there are $3^4 + 3^2 + 3^0 + 3^0 = 92$.

The second p-decision can be delegated to q-decision 1, 2, 3, or 4. There are $1^4 + 1^2 + 1^0 + 1^0 = 4$ ways to do this; similarly there are four ways to delegate the third p-decision to a q-decision.

In total, there are $92*4*4 = 1472$ morphisms $p \to q$, just twenty less than the number of years between Jesus's famous year and Columbus's famous year. Coincidence?

Exercise 4.67.

1. Draw the poly's (unions of corollas as in Eq. (4.2)) associated to $p \coloneqq y^3 + y + 1$, $q \coloneqq y^2 + y^2 + 2$, and $r \coloneqq y^3$.
2. Give an example of a morphism $p \to q$ and draw it as we did in Example 4.66.
3. Explain your morphism as a "delegation" in the sense of Section 4.2.3.
4. Explain in those terms why there can't be any morphisms $p \to r$. ◊

Exercise 4.68. 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 the associated poly as well as y^2.
2. Count all the maps $y^2 \to p$.
3. Is it equal to $p(2)$? ◊

Example 4.69 (Derivatives). We won't have much opportunity to use the following, but it may be interesting. The *derivative* of a polynomial p, denoted $\dot{p}$, is defined as follows:

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

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

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

Up to isomorphism $p \cong y^3 + 2y$ and $\dot{p} \cong 3y^2 + 2$.

A morphism $f : p \to \dot{q}$ can be interpreted in terms of delegations from p to q such that each position in p explicitly selects a q-option to be avoided. More precisely, for

each $i \in p(1)$ we have $f_1(i) = (j, d) \in \sum_{j \in q(1)} q[j]$, i.e. a choice of q-position j, as usual, together with a chosen option $d \in q[j]$. Then every q-option *other than* d is passed back to an option in $p[i]$.

Exercise 4.70. The derivative is not very well-behaved category-theoretically. However, it is intriguing.

1. Is there always a canonical map $p \to \dot{p}$?
2. Is there always a canonical map $\dot{p} \to p$?
3. If given a map $p \to q$, does one get a map $\dot{p} \to \dot{q}$?
4. What is the relationship between $\dot{p}(1)$ and the total number of leaves in p?
5. In terms of problems and solutions, explain what a map $p \to \dot{q}$ would say.
6. Read the formula for $\otimes$ and $[-,-]$ in (4.88) and (4.156) and find a formula for a map $p \otimes [p, y] \to p'$ that works for any $p \in$ **Poly**.
7. When talking to someone who explains maps $p \to \dot{q}$ in terms of "avoiding options", how might you describe what is modeled by a map $py \to q$?

◊

Working with $\sum$ and $\prod$ formulas. Here's how to understand and work with a set that involves $\sum$'s and $\prod$'s, e.g. something like

$$\sum_{i \in I} \prod_{j \in J(i)} \sum_{k \in K(i,j)} X(i, j, k). \tag{4.71}$$

To understand it, you ask "what is involved in choosing an element of this set?"

The answer given by the formula (4.71) begins by saying "To choose an element of (4.71), first..." and then you begin pronouncing the formula. Every time you see $\sum_{i \in I} \ldots$, say "choose an element $i \in I$; then, ...". Every time you see $\prod_{j \in J}$, say "for each $j \in J$, ...". At the very end when you see some X, say "conclude by choosing an element of X".

So the formula in (4.71) would be pronounced as follows:

> To choose an element of (4.71), first choose an element $i \in I$. Then, for every $j \in J(i)$, choose an element $k \in K(i, j)$; then, conclude by choosing 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 these sets $K(i, j)$ can be different for different i, j.

Example 4.72 (Notation for $\sum \prod$ stuff). Here we give notation for the elements of a set involving $\sum$'s and $\prod$'s such as that in (4.71).

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 $K(2, j') \coloneqq \{k'\}$, and let $X(i, j, k) = \{x, y\}$ for all i, j, k. Now the formula

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

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

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

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

Exercise 4.73. With I, J, K, and X as in Example 4.72, consider the set

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

1. Pronounce it using the "To choose an element" formulation just below (4.71).
2. How many elements are in it?
3. Write out three of them in the style of Example 4.72. ◊

The following is sometimes called the *type-theoretic axiom of choice* or the *completely distributive property*, in this case of **Set**. It is almost trivial, once you understand what it's saying; for example once the statement is written in agda, its proof is one short line of agda code.

Proposition 4.74 (Pushing $\prod$ past $\sum$)**.** For any set I, collection of sets $\{J(i)\}_{i \in I}$, and collection of sets $\{X(i, j)\}_{i \in I, j \in J(i)}$, we have a bijection

$$\sum_{j \in \prod_{i \in I} J(i)} \prod_{i \in I} X(i, j(i)) \cong \prod_{i \in I} \sum_{j \in J(i)} X(i, j). \tag{4.75}$$

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

First, let's go from left to right. An element of the set on the left is a pair of a function $j \in \prod_{i \in I} J(i)$ together with a function $f \in \prod_{i \in I} X(i, j(i))$. We then get an element of the set on the right as follows:

$$i \mapsto (j(i), f(i)).$$

Now, let's go right to left. An element of the set on the right is a function $f \in \prod_{i \in I} \sum_{j \in J(i)} X(i, j)$. We can then form the following pair in the left hand set:

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

where $\pi_1 f(i)$ is the first component of the pair $f(i)$ and $\pi_2 f(i)$ is the second component.

Now, we just need to check that a round trip takes us back where we were. If we start on the left, our round trip gives us the pair

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

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

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

But again, since $f(i)$ is the pairing of its components $\pi_1(f(i))$ and $\pi_2(f(i))$, we're back where we started.

□

Exercise 4.76. Use Proposition 4.74 to give another proof of Lemma 4.61. (Hint: express the left hand side as a dependent sum.) ◊

Below, e.g. in Exercise 4.77, you'll often see alternating sums and products; using Eq. (4.75), you can always put it into sums-then-products ("disjunctive normal") form.

When $J(i) = J$ does not depend on $i \in I$, the formula in (4.75) is much easier:

$$\prod_{i \in I} \sum_{j \in J} X(i,j) \cong \sum_{j : I \to J} \prod_{i \in I} X(i,j).$$

Exercise 4.77. Let p, q be polynomials as in Proposition 4.62. Recall from our main formula (4.63) that there is an isomorphism between polynomia$\mathbf{Poly}(p,q) \cong \prod_{i \in p(1)} \sum_{j \in q(1)} p[i]^{q[j]}$.

1. Is it true that the following are isomorphic?

$$\mathbf{Poly}(p,q) \cong^? \prod_{i \in p(1)} \sum_{j \in q(1)} \prod_{d \in q[j]} \sum_{c \in p[i]} 1 \tag{4.78}$$

2. Is it true that there is an isomorphism

$$\mathbf{Poly}(p,q) \cong^? \sum_{f_1 : p(1) \to q(1)} \prod_{j \in q(1)} \mathbf{Set}\Big(q[j], \prod_{\{i \in p(1) | f_1(i) = j\}} p[i]\Big). \tag{4.79}$$

3. If the answer to #2 is "yes", then say how any element of the right-hand side gives a way of delegating decisions from p to q. If "no", give intuition for why the two sets are not isomorphic. ◊

Back to maps of polynomials After that interlude, you are hopefully more comfortable with our main formula Eq. (4.63) describing the set of morphisms between two arbitrary polynomial functors as dependent lenss. Let's practice.

Example 4.80 (Constants are sent to constants). Any natural transformation $p \to q$ must send constants in p to constants in q. So if p has constant terms and q doesn't, e.g. if $p = y^2 + 1$ and $q = y^3$, then there are no maps $p \to q$.

Indeed, let's more generally say $p \coloneqq \sum_{i \in p(1)} y^{p[i]}$ and $q \coloneqq \sum_{j \in q(1)} y^{q[j]}$ are polynomials, and that we have a map $p \to q$ corresponding to a dependent lens $\left(\begin{smallmatrix} p[i] \\ i \in p(1) \end{smallmatrix}\right) \leftrightarrows \left(\begin{smallmatrix} q[j] \\ j \in q(1) \end{smallmatrix}\right)$. If there is some $i_0 \in p(1)$ such that $p[i_0] = 0$, then the formula says that for i_0 our map has chosen a $j \in q(1)$ and a function $q[j] \to p[i]$. But in order for such a function to exist, if $p[i]$ is empty, $q[j]$ must be too.

Exercise 4.81. For each of the following p, q, count the number of natural transformations $p \to q$:

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

Exercise 4.82 (A functor **Top** $\to$ **Poly**). This exercise is for those who know topological spaces and the maps between them. It will not be used again in this book.

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

4.2.4 Prepare for dynamics

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

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

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

Proposition 4.83. The category **Poly** has finite products.

Proof. We will see that 1 is a terminal object and that the product of p and q in **Poly** is the usual product of p and q as polynomials. That is, if $p \coloneqq \sum_{i \in p(1)} y^{p[i]}$ and $q \coloneqq \sum_{j \in q(1)} y^{q[j]}$ are in standard notation, then

$$p \times q \cong \sum_{i \in p(1)} \sum_{j \in q(1)} y^{p[i]+q[j]}. \tag{4.84}$$

We leave the proof as an exercise; see Exercise 4.85. □

Exercise 4.85. Use Eq. (4.63) to prove Proposition 4.83 using dependent lenses. In particular:

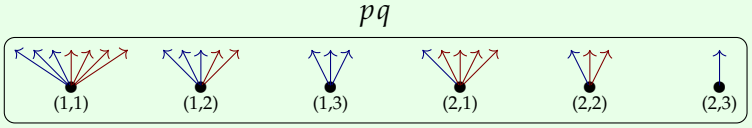
1. Prove that 1 is terminal in **Poly**.
2. Prove that for polynomials p, q, their product is given by the formula in (4.84). ◊

We will begin writing pq rather than $p \times q$:

$$pq \coloneqq p \times q \tag{Notation}$$

Example 4.86. We can draw the product of two polynomials in terms of their associated poly's. Let $p \coloneqq y^3 + y$ and $q \coloneqq y^4 + y^2 + 1$.

Then $pq \cong y^7 + 2y^5 + 2y^3 + y$. In pictures, we take all pairs of decisions, and for each pair we take the union the options.



Parallel product There is a closely related monoidal structure on **Poly** that will be useful for putting dynamical systems in parallel and then wiring them together.

Definition 4.87. Let p and q be polynomials. When they are expressed in standard notation, their *parallel product*, denoted $p \otimes q$ is given by the formula:

$$p \otimes q \cong \sum_{i \in p(1)} \sum_{j \in q(1)} y^{p[i] \times q[j]}. \tag{4.88}$$

On arenas, this is defined by

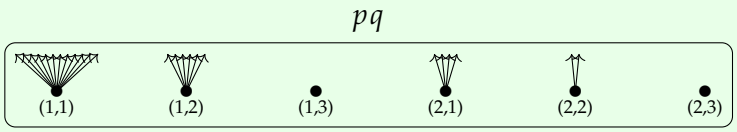
$$\begin{pmatrix} p[i] \\ i \in p(1) \end{pmatrix} \otimes \begin{pmatrix} q[j] \\ j \in q(1) \end{pmatrix} := \begin{pmatrix} p[i] \times q[j] \\ (i,j) \in (p \times q)(1) \end{pmatrix}. \tag{4.89}$$

One should compare this with the formula for product of polynomials shown in (4.84). The difference is that parallel product multiplies exponents where regular product adds them.

Remark 4.90. The reason we call this the parallel product is because it generalizes the parallel product for lenses in Section 1.3.1.

Example 4.91. We can draw the product of two polynomials in terms of their associated poly's. Let $p := y^3 + y$ and $q := y^4 + y^2 + 1$.

Then $p \otimes q \cong y^{12} + y^6 + y^4 + y^2 + 2$. In pictures, we take all pairs of decisions, and for each pair we take the product of the options.



Exercise 4.92. What is the parallel product of monomials

$$A_1 y^{B_1} \otimes A_2 y^{B_2}?$$

◊

Exercise 4.93. Let $p := y^3 + y$ and $q := 2y^4$. Draw the following arenas using corollas:

1. Draw p and q as arenas.

2. Draw $pq = p \times q$ as an arena.
3. Draw $p \otimes q$ as an arena. ◊

Exercise 4.94. Consider the polynomials $p \coloneqq 2y^2 + 3y$ and $q \coloneqq y^4 + 3y^3$.

1. What is $p \times q$?
2. What is $p \otimes q$?
3. What is the product of the following purely formal expression we'll see *only this once!*:
$$(2{\cdot}2^y + 3{\cdot}1^y + 1) \cdot (1{\cdot}4^y + 3{\cdot}3^y + 2)$$
These are called Dirichlet series.
4. Do you see a connection between what the last two parts? If so, would you say that it justifies the name "Dirichlet product" as another name for the parallel product $\otimes$?

◊

Exercise 4.95. What is $(3y^5 + 6y^2) \otimes 4$? Hint: $4 = 4y^0$. ◊

Exercise 4.96. 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 $(p \otimes q) \cong (q \otimes p)$. ◊

In Exercise 4.96 we have gone most of the way to proving that $(\mathbf{Poly}, \otimes, y)$ is a symmetric monoidal category.

Proposition 4.97. The category **Poly** has a symmetric monoidal structure $(y, \otimes)$ where $\otimes$ is the parellel product from Definition 4.87.

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

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

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

$$(f \otimes g)^\sharp_{(i,j)}(d, e) \coloneqq \big(f_i^\sharp(d), g_j^\sharp(e)\big).$$

We have not proven all the coherences between the various isomorphisms, but we ask the reader to take it on trust or to check it for themselves. □

Exercise 4.98.

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 $p(1) \times q(1)$? ◊

Exercise 4.99. 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 quadradic polynomials.
5. The set $\{Ay^B \mid A, B \in \mathbf{Set}\}$ of monomials.
6. The set $\{Sy^S \mid S \in \mathbf{Set}\}$ of systematic polynomials.
7. The set $\{p \in \mathbf{Poly} \mid p(1) \text{ is finite }\}$.

What is the smallest class of polynomials that's closed under $\otimes$ and contains y? ◊

Exercise 4.100. Show that for any $p_1, p_2, q \in \mathbf{Poly}$ there is an isomorphism

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

Proposition 4.101. For any monoidal structure $(I, \odot)$ on **Set**, there is a corresponding monoidal structure on **Poly** with unit y^I. It is given by Day convolution, and it distributes over $+$.

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

Proof. Day convolution always takes a monoidal structure on **Set** and returns one on functors **Set** $\to$ **Set**; the issue is whether **Poly** is closed under the resulting monoidal product. We will show that it is, roughly because every polynomial is a coproduct of representables.

** □

Exercise 4.102.

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 product on **Poly**. What is $(y^3 + y) \odot (2y^2 + 2)$? ◊

⊗-monoids in Poly

Definition 4.103. A *⊗-monoid in **Poly*** consists of a polynomial m, a map $(*)\colon m \otimes m \to m$, and a position $e\colon y \to m$ that form a monoid in $(\mathbf{Poly}, y, \otimes)$.

These have a kind of swarm-like semantics, making individuals able to summarize the positions of a group of subordinates, as well as distribute instructions to those subordinates in a coherent way.

Exercise 4.104. Explain how $\odot$ has the "swarm-like" semantics described above. ◊

Example 4.105 (Monoids in **Set** give linear ⊗-monoids). If $(M, e, (*))$ is a monoid in $(\mathbf{Set}, 1, \times)$ then the linear polynomial My carries a corresponding ⊗-monoid, roughly because $Ay \otimes By \cong (A \times B)y$. This construction is functorial and fully faithful: a map between monoids in **Set** can be identified with a map between ⊗-monoids.

Example 4.106 (Comonoids in **Set** give representable ⊗-monoids). Every comonoid (S, ϵ, δ) in $(\mathbf{Set}, 1, \times)$ gives rise to a ⊗-monoid structure on y^S. This construction is again (contravariantly) functorial and fully faithful.

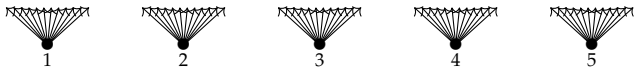
Example 4.107. Let $m \coloneqq \mathbb{R}_{\geq 0} y^{\mathbb{R}_{\geq 0}}$. Take $e = 0$ and $(a, b) \mapsto (a + b, t \mapsto (\frac{at}{a+b}, \frac{bt}{a+b}))$. This forms a ⊗-monoid.

Proposition 4.108. If m, n are the carriers of ⊗-monoids, then $m \otimes n$ naturally also carries a ⊗-monoid structure.

Proposition 4.109 (Free ⊗-monoid)**.** The forgetful functor from ⊗-monoids to **Poly** has a left adjoint.

Proof. The carrier p' of the free ⊗-monoid on p has positions $p'(1) \cong \mathsf{List}(p(1))$, i.e. lists of positions in p. Given a position $\ell = (i_1, \ldots, i_n)$, the direction-set there is given by $\prod_{j \in \mathsf{n}} p[j]$. ** □

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



The formula for morphisms between these is particularly simple:

$$\mathbf{Poly}\left(A_1y^{B_1},\, A_2y^{B_2}\right) \cong \mathbf{Set}(A_1, A_2) \times \mathbf{Set}(A_1 \times B_2, B_1)$$

It says that to give a morphism from one monomial to another, you just need to give two functions. Let's rewrite it to make those two functions explicit:

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

Ordinarily, the $f^\sharp$ is more involved in that its type depends on f_1, but for monomials every decision has the same number of options, so to speak.

The monomials in **Poly** and the morphisms between them forms a full subcategory of **Poly**, and it has been called the *the category of bimorphic lenses* [**hedges2018limits**]. It comes up in functional programming. The morphisms $A_1y^{B_1} \to A_2y^{B_2}$ break up into two functions which people give special names:

$$\begin{aligned} &\text{get} \colon A_1 \to A_2 \\ &\text{set} \colon A_1 \times B_2 \to B_1 \end{aligned} \tag{4.110}$$

The idea is that each A-decision "gets" a B-decision, and every option there in B "sets" an option back in A

We will use the term *lens* when we want to remind the reader that morphisms between monomials are much simpler than those between arbitrary polynomials.

Remark 4.111. Consider lenses between polynomials of the form Sy^S. In terms of arenas, the set of options for each decision $s \in S$ is again just the set S of decisions. So decisions are all about "which $s \in S$ you're at, and which one you want to be at next".

A lens, i.e. a polynomial map $(\text{get}, \text{set}) \colon Sy^S \to Ty^T$ is as usual a way to delegate S-decisions to T-decisions. Among them, some lenses are considered to be "very well behaved" because they give T a more sensible form of control. Namely, they are the ones that satisfy the following for all $s \in S$ and $t, t' \in T$:

$$\text{get}(\text{set}(s,t)) = t \qquad \text{set}(s, \text{get}(s)) = s \qquad \text{set}(\text{set}(s,t), t') = \text{set}(s, t')$$

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

4.3 Dynamical systems using Poly

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

4.3.1 Moore machines

Remember Moore machines from Definition 1.20?

Definition 4.112. If A, B, and A are sets, an (A, B)-Moore machine with states S consists of two functions

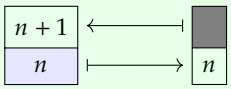
$$\text{expose}\colon S \to B$$
$$\text{update}\colon S \times A \to S$$

Look familiar? It's easy to see that an (A, B) Moore machine with states S is just a map of polynomials

$$Sy^S \to By^A$$

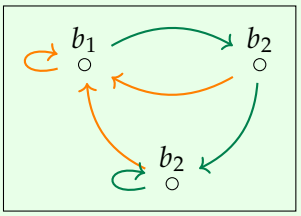
or to say it another way, a lens from $(S, S) \to (A, B)$ in the sense of Eq. (4.110).

Example 4.113. There is a dynamical system that takes unchanging input and produces as output the sequence of natural numbers $0, 1, 2, 3, \ldots$. It is a Moore machine of type $(1, \mathbb{N})$ with states $\mathbb{N}$. The polynomial map $\mathbb{N}y^{\mathbb{N}} \to \mathbb{N}y$ is given by identity on positions and $n \mapsto n + 1$ on directions



We can generalize both Sy^S and By^A in Definition 4.112, though the latter is much simpler and we'll spend much of the section on it. After some examples, we'll briefly preview something we're going to spend a lot of time later, namely what's special about Sy^S; then we'll discuss how to model regular languages using Moore machines and how products of polynomials allow multiple interfaces to act on the same system.

Example 4.114. Here's a picture of a Moore machine with $S := 3$ states:



Each state is labeled by its exposed value, i.e. element of $B := \{b_1, b_2\}$. Each state has two outgoing arrows, one orange and one green, so $A := \{\text{orange}, \text{green}\}$.

You can imagine barking "orange! orange! green! orange!" etc. at this machine, and it running through states.

Example 4.115. Here's a(n infinite) Moore machine with states $\mathbb{R}^2$:

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

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

$$\begin{array}{cc} \mathbb{R}^2 \xrightarrow{\text{get}} \mathbb{R}^2 & \mathbb{R}^2 \times [0,1] \times [0,2\pi] \xrightarrow{\text{put}} \mathbb{R}^2 \\ (a,b) \mapsto (a,b) & (a,b,r,\theta) \mapsto (a + r\cos\theta, b + r\sin\theta) \end{array}$$

Example 4.116 (From functions to Moore machines). For any function $f : A \to B$ there is a corresponding (A,B)-Moore machine with states A that takes in a stream of A's and outputs the stream of B's obtained by applying f. It is given by the map $(f, f^\sharp) : Ay^A \to By^A$ that on positions is f, and for each $a \in A$ the on-directions map $f^\sharp_a : A \to A$ is the identity on A.

Exercise 4.117. For any function $f : A \to B$ there is a corresponding (A,B)-Moore machine with states B that takes in a stream of A's and outputs the stream of B's obtained by applying f. What is it? ◊

Exercise 4.118. Find $A, B \in$ **Set** such that the following can be identified with a morphism $Sy^S \to By^A$:

1. a *discrete dynamical system*, i.e. a set S and a function $S \to S$.
2. a *magma*, i.e. a set S and a function $S \times S \to S$.
3. a set S and a subset $S' \subseteq S$. ◊

Exercise 4.119. Consider the Moore machine in Example 4.115, 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 value between 0 and 10. Make the robot lose health whenever the x-coordinate of its position is negative. Use its health h as a multiplier, allowing it to move a distance of hr given an input of r. ◊

Exercise 4.120. Let's say a file of length n is a function $f : \mathsf{n} \to \mathsf{ascii}$, where $\mathsf{ascii} := 256$. Suppose given such a file and refer to elements of $\mathsf{n} = \{1, \dots, n\}$ as positions. Make a robot (Moore machine) whose output type is ascii and whose input type is

$$\{(s,t) \mid 1 \le s \le t \le n\} + \{\text{continue}\}$$

Given an input, if it is of the form (s, t) then the robot goes to position s in the file and begins reading (assuming it receives the "continue" signal) one character at a time, until it reaches position t. Then it continually outputs "done" until it receives another (s, t) pair. ◊

When we get to generalized Moore machines, we will be able to let the robot "close its port", so that while it can't receive signals while it's busy reading; see Example 4.136.

Exercise 4.121 (Tape of a Turing machine). A Turing machine has a tape. The tape has a position for each integer, and each position holds a value $v \in V = \{0, 1, -\}$ of 0,1, or blank. At any given time the tape not only holds this function $f : \mathbb{Z} \to V$ from positions to values, but also a distinguished choice $c \in \mathbb{Z}$ of "current" position. Thus the set of states of the tape is $T := V^{\mathbb{Z}} \times \mathbb{Z}$.

The Turing machine interacts with the tape by asking for the value at the current position, an element of V, and by telling it to change the value there as well as whether to move left or right. Thus the output of the tape is V and the input is $V \times \{L, R\}$.

1. Give the form of the tape as a Moore machine, i.e. map of polynomials $t : Sy^S \to p$ for appropriate $S \in S$ and $p \in$ **Poly**.
2. Write down the specific t that makes it act like a tape as specified above. ◊

The polynomial Sy^S as a comonad on Set. A *comonad* on **Set** is a functor $F :$ **Set** $\to$ **Set**, equipped with two natural transformations $\epsilon : F \to \mathrm{id}$ and $\delta : F \to F \circ F$, satisfying three equations. We don't need this now, so we won't get into it here. But we will note that every comonad comes from an adjunction, and the adjunction corresponding to Sy^S is

$$\mathbf{Set} \underset{\_^S}{\overset{-\times S}{\underset{\Rightarrow}{\rightleftarrows}}} \mathbf{Set}$$

the "curry/uncurry" adjunction. In functional programming, the comonad Sy^S is called the *state comonad*,[3] and the elements of S are called states. It is no coincidence that we also refer to elements of S as states.

Again, we will be *very* interested in polynomial comonads later—as mentioned in Proposition 4.18 they are exactly categories!!—but for now we move on to things we can use right away in our story about dynamical systems.[4]

Regular languages Regular languages are very important in computer science. One way to express what they are is to say that they are exactly the languages recognizable by a deterministic finite state automaton. What is that?

[3] The comonad Sy^S is sometimes called the *store* comonad.

[4] If you're curious what category the comonad Sy^S corresponds to, it's the one with object set S and a unique morphism $s_1 \to s_2$ for every pair of objects $s_1, s_2 \in S$.

Definition 4.122. A *deterministic finite state automaton* consists of

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

Proposition 4.123. A deterministic finite state automaton can be identified with a pair of maps

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

Proof. A map $y \to Sy^S$ can be identified with an element $s_0 \in S$. A map $Sy^S \to 2y^A$ consists of a function $S \to 2$—which can be identified with a subset of S—together with a function $S \times A \to S$, which is the rest of the required structure. □

Products: two interfaces operating on the same system. One thing we can use right away in our thinking about dynamical systems is products of polynomials. The universal property of products says that for any polynomials s, p_1, p_2 and maps $s \to p_1$ and $s \to p_2$ there is a unique map $s \to p_1 p_2$. In the context of Moore machines, we have the following.

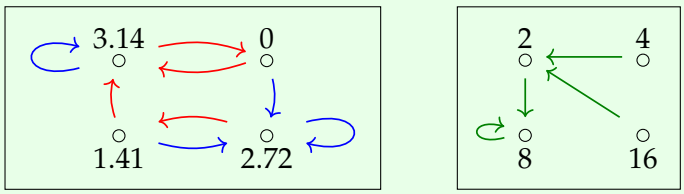
Proposition 4.124. Suppose given an (A_1, B_1)-Moore machine and an (A_2, B_2)-Moore machine, each with state set S. Then there is an induced (A_1+A_2, B_1B_2)-Moore machine, again with state set S.

Proof. We are given maps of polynomials $Sy^S \to B_1 y^{A_1}$ and $Sy^S \to B_2 y^{A_2}$. Hence by the universal property of products we have a map

$$Sy^S \to (B_1 y^{A_1}) \times (B_2 y^{A_2}) \cong (B_1 B_2) y^{A_1 + A_2},$$

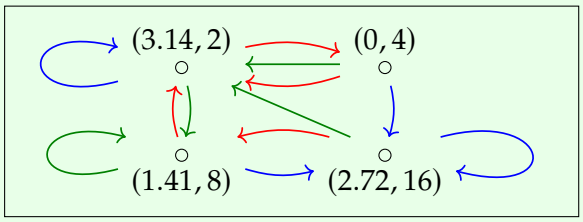
as desired. □

Example 4.125. Consider two four-state dynamical systems $4y^4 \to \mathbb{R}y^{\{r,b\}}$ and $4y^4 \to \mathbb{R}y^{\{g\}}$, each of which gives outputs in $\mathbb{R}$; we think of r, b, g as red, blue, and green, respectively. We can draw such morphisms as labeled transition systems, e.g.



The universal property of products provides a unique way to put these systems

together to obtain a morphism $4y^4 \to (\mathbb{R}y^{\{r,b\}} \times \mathbb{R}y^{\{g\}}) = (\mathbb{R}^2)y^{\{r,b,g\}}$. With the examples above, it looks like this:



In other words, if there are two or more interfaces that drive a single set of states, we can combine them.

Exercise 4.126 (Toward event-based systems). Let $f : Sy^S \to By^A$ be a Moore machine. 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 machine not to do anything. That is, rather than needing to press a button in A at each time step, we want to be able to *not* press any button, in which case the machine just stays where it is. We want a new machine $f' : Sy^S \to p$ that has this behavior; what is p and what is f'? ◊

4.3.2 Dependent Systems

Everything we've done above was for interfaces of the form By^A, i.e. for monomials. But the theory works just as well for an arbitrary p, a sum of monomials.

Definition 4.127 (Dependent systems)**.** A *dependent system* is a map of polynomials

$$Sy^S \to p$$

for some $S \in$ **Set** and $p \in$ **Poly**. The set S is called the set of *states* and the polynomial p is called the *(poly-) interface*.

We could also call these *generalized Moore machines*, since Moore machines were seen to be given by the special case $p = By^A$ for sets A, B. A polynomial might be $p = B_1y^{A_1} + \cdots + B_ky^{A_k}$ (though it doesn't have to be finite), so the output is an element of the coproduct $B_1 + \cdots + B_k$, and the choice of output determines what sort of input you get. What kind of output is that?

We now begin to think of outputs not only as outward expressions, but as positions that one takes within ones arena, terminology we've been using all along. If the arena is your body, this would be positions of your body. 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, all talking and gesturing 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 a sensory organ, a hand outstretched, an eye open or closed f.

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.

But however you think of dependent systems, we need to get a feel for how they work. Let's start with something familiar.

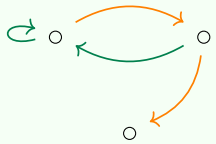
Example 4.128. Recall regular languages from Definition 4.122. Often one does not want to "keep going" after recognizing a word in ones language. For that, rather than use a map $Sy^S \to 2y^A$, we could use a map

$$(f_1, f^\sharp)\colon Sy^S \to y^A + 1$$

To give such a map, one in particular provides a function $f_1\colon S \to 2$; here we are thinking of 2 as the set of positions in $y^A + 1$. A function f_1 sends some elements of S to 1 and sends others to 2; those that are sent to 2 are said to be "accepted". This is captured by the fact that there are no options in the term 1, no inputs available there. In other words, the function $f^\sharp$ is trivial there.

On the other hand, $f_s^\sharp$ is not trivial on those elements $s \in S$ for which $f_1(s) = 1$. They must come equipped with a function $f_s^\sharp\colon A \to S$, saying how the machine updates on each element of A, starting at state s. Again, this is in line with the way state machines encode regular languages.

Exercise 4.129. Consider the deterministic finite state automaton shown below



The state without any outgoing arrows is the "accept" state for a regular language, and the left-most state is the start state. Answer the following questions, in keeping with the notation from Example 4.128.

1. What is S?
2. What is A?
3. In terms of regular languages, what is the alphabet here?
4. Specify the morphism of polynomials $Sy^S \to y^A + 1$.
5. Name a word that is accepted by this machine.
6. Name a word that is not accepted by this machine. ◊

Example 4.130 (Adding a pause button). Given any dependent dynamical system $f\colon Sy^S \to p$, we can "add a pause button", meaning that for any state (and any position), we add an input that keeps the state where it is.

To do this, note that we have a map $\epsilon\colon Sy^S \to y$ given by identity (see **??**). By the universal property of products, we can pair f and ϵ to get a map $(f,\epsilon)\colon Sy^S \to py \cong p \times y$. This process is actually universal in a way we'll find important later. Indeed, it's called *copointing*; see **??**.

Example 4.131. Suppose given a dependent dynamical system f that sometimes outputs an element of A and sometimes outputs only the hum. That is, its interface is $Ay + y$. What if we want to turn that into a dynamical system that always outputs an A by constantly repeating the last thing output by f. How do we construct it?

The output to our whole system is supposed to be Ay, and we already have $f\colon Sy^S \to Ay + y$. It will be enough to construct a map $g\colon (Ay + y) \otimes Ay^A \to Ay$, because then we can use

$$(SAy^{SA}) \cong (Sy^S) \otimes (Ay^A) \xrightarrow{f\otimes \mathrm{id}} (Ay + y) \xrightarrow{g} Ay$$

as our overall dynamical system. Since $\otimes$ distributes over $+$, we need to give $Ay \otimes Ay^A \to Ay$ and $y \otimes Ay^A \to Ay$. For the latter we use the identity, which we called $\epsilon\colon Ay^A \to y$ and for the former we use

$$Ay \otimes Ay^A \cong (Ay \otimes y^A) \otimes Ay \xrightarrow{\epsilon\otimes \mathrm{id}} y \otimes Ay$$

This accomplishes what we wanted.

Example 4.132 (Inputting a start state). Suppose you have a closed system $f^\sharp\colon Sy^S \to y$. The modeler can choose a start state $y \to Sy^S$, but what if we want some other system to choose the start state? We haven't gotten to wiring diagrams yet, but the idea is to create a system that starts as not-closed—accepting as input a state $s \in S$—and then dives into its closed loop with that start state.

Let $S' \coloneqq S + 1$, so that the start state $y \to S'y^{S'}$ now is canonical: it's the new 1. We also have a canonical inclusion $S \xrightarrow{i} S'$. We will give a morphism

$$S'y^{S'} \to y + y^S$$

that starts out with its outer box in the mode y^S of accepting an S-input, and then moves to the mode y so that it is a closed system forever after.

To give a morphism $S'y^{S'} \to y + y^S$, it is sufficient to give two morphisms: $Sy^{S'} \to y$ and $y^{S'} \to y^S$. The first is equivalent to a function $S \to S'$ and we take the map $S \xrightarrow{f^\sharp} S \xrightarrow{i} S'$; this means that whenever we want to update the state from a state in S

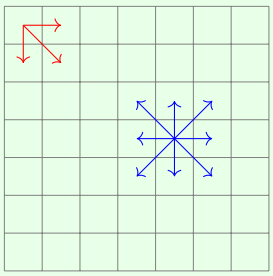
we'll just do whatever our original closed system did. The second is also equivalent to a function $S \to S'$ and we use i; this means that whatever state is input at the beginning will be what we take as our first noncanonical state.

Exercise 4.133. Find what you think is an interesting generalization of deterministic finite state automata that you can model using generalized Moore machines. ◊

Example 4.134. Choose $n \in \mathbb{N}$, a *grid size*, and for each $i \in \mathsf{n} = \{1, \dots, n\}$ let $d(i)$ be the set

$$d(i) := \begin{cases} \{0,1\} & \text{if } i = 1 \\ \{-1,0\} & \text{if } i = n \\ \{-1,0,1\} & \text{if } 1 < i < n \end{cases}$$

So $d(i)$ is the set of directions someone could move (or not move) if at position i.



Then a generalized Moore machine of the form $Sy^S \to p$, where

$$p = \sum_{(i,j) \in \mathsf{n} \times \mathsf{n}} y^{d(i) \times d(j)},$$

is one that has more movement options when it is in the center of the grid than when it is on the sides or corners.

Exercise 4.135. Add to Example 4.134 as follows.

1. Redefine p so that at each grid value, the robot can receive not only the set of directions it can move in but also a "reward value" $r \in \mathbb{R}$.
2. Define the set S of robot states so that an element $s \in S$ includes both the robot's position and a list of all reward values so far.
3. Define a morphism of polynomials $Sy^S \to p$ in a way that respects positions and properly updates the robots list of rewards, but otherwise does anything you want. ◊

Example 4.136. In Exercise 4.120 one is tasked with making a file reader FileReader, where a file is a function $f : \mathsf{n} \to \mathsf{ascii}$. Now we take that same idea but make the robot have a different interface when it is in read-mode: namely, one where it cannot take in signals.

Let $\mathsf{State}_{\mathsf{FileReader}} := \{(s,t) \mid 1 \leq s \leq t \leq n\}$ consist of a current position s and a terminal position t. For our interface, we'll have two modes, each of which exposes an ascii character:

$$\mathsf{Out}_{\mathsf{FileReader}} = \langle \mathrm{Accepting}(c), \mathrm{Busy}(c) \mid c \in \mathsf{ascii} \rangle$$

For our input, we need a family $\mathsf{In}_{\mathsf{FileReader}} : \mathsf{Out}_{\mathsf{FileReader}} \to \mathbf{Set}$. We'll define this by cases:

$$\mathsf{In}_{\mathsf{FileReader}}(\mathrm{Accepting}(c)) = \mathsf{State}_{\mathsf{FileReader}},$$
$$\mathsf{In}_{\mathsf{FileReader}}(\mathrm{Busy}(c)) = 1.$$

Our file reader will be Accepting if its current position is the terminal position; otherwise, it will be Busy. In either case, it will expose the ascii character at the current position.

$$\mathrm{expose}_{\mathsf{FileReader}}(s,t) = \begin{cases} \mathrm{Accepting}(f(s)) & \text{if } s = t \\ \mathrm{Busy}((f(s))) \end{cases}$$

While the file reader is Busy, it will step forward through the file. When it is Accepting, it will set its new current and terminal position to be the input.

$$\mathrm{update}_{\mathsf{FileReader}}(s,t) = \begin{cases} \_ \mapsto (s+1,t) & \text{if } \mathrm{expose}_{\mathsf{FileReader}}(s,t) \text{ is Busy} \\ (s',t') \mapsto (s',t') & \text{if } \mathrm{expose}_{\mathsf{FileReader}}(s,t) \text{ is Accepting} \end{cases}$$

Let $A := \{(s,t) \mid 1 \leq s \leq t \leq n\}$, and let $p := \mathsf{ascii} \cdot y^A + \mathsf{ascii} \cdot y^1$; we construct a morphism in **Poly**

$$(r_1, r^\sharp) \colon Ay^A \to \mathsf{ascii} \cdot y^A + \mathsf{ascii} \cdot y^1$$

as follows.

$$A + B = \langle \mathrm{inl}(a), \mathrm{inr}(b) \mid a \in A, b \in B \rangle$$

$$A +_C B = \langle \mathrm{inl}(a), \mathrm{inr}(b) \mid a \in A, b \in B, \forall c \in C . \mathrm{inl}(f(c)) = \mathrm{inr}(g(c)) \rangle$$

On positions define

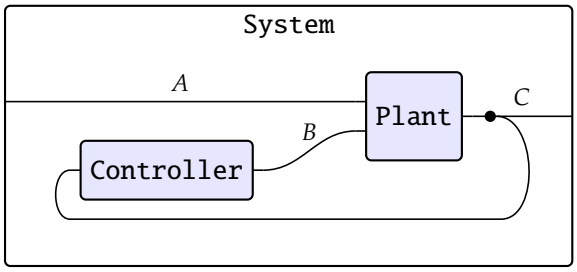
$$r_1(s,t) := \begin{cases} \mathtt{inl}\ f(s) & \text{if } s = t \\ \mathtt{inr}\ f(s) & \text{if } s < t \end{cases}$$

$$r^\sharp_{(t,t)}(s',t') \coloneqq (s',t') \qquad r^\sharp_{(s,t)}(1) \coloneqq (s+1,t)$$

Exercise 4.137. Make a file reader that acts like that in Example 4.136, except that it only emits output $o \in \text{ascii}$ when $o = 100$. ◊

4.3.3 Wiring diagrams

We want our dynamical systems to interact with each other.



(4.138)

In this picture 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 (4.138): the controller, the plant, and the system. Each has inputs and outputs, and so we can consider the interface as a monomial.

$$\texttt{Plant} = Cy^{AB} \qquad \texttt{Controller} = By^{C} \qquad \texttt{System} = Cy^{A}. \tag{4.139}$$

The wiring diagram itself is a morphism in **Poly** of the form

$$w\colon \texttt{Plant} \otimes \texttt{Controller} \to \texttt{System}$$

Since everything involved is a monomial—the parallel product of monomials is a monomial—the whole wiring diagram w is a lens $CBy^{ABC} \to Cy^{A}$. This morphism says how wires are feeding from outputs to inputs. Like all lenses, it consists of two functions

$$\text{get}\colon CB \to C \qquad \text{and} \qquad \text{put}\colon CBA \to ABC$$

The first says "inside the system you have boxes outputting values of type C and B. The system needs to produce an output of type B; how shall I obtain it?" The second says "the system is providing an input value of type A, and inside the system you have boxes outputting values of type C and B. These boxes need input values of type A, B, and C; how shall I obtain them?" The answer of course is that *get* is given by projection $(c,b) \mapsto c$ and *put* is given by a permutation $(c,b,a) \mapsto (a,b,c)$. The wiring diagram is a picture that tells us which maps to use.

Exercise 4.140.

1. Make a new wiring diagram like (4.138) except where the controller also receives information of type A' from the outside world.
2. What are the monomials in your diagram (replacing (4.139))?
3. What is the morphism of polynomials corresponding to this diagram? ◊

Now suppose given a dynamical system in each inner box:

$$Sy^S \xrightarrow{f} \texttt{Plant} \qquad \text{and} \qquad Ty^T \xrightarrow{g} \texttt{Controller}$$

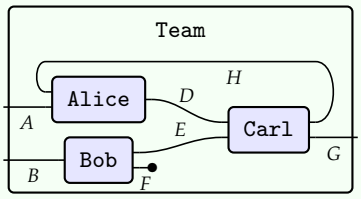
Then since $\otimes$ is a monoidal product on **Poly** (see Proposition 4.97), we get a map

$$Sy^S \otimes Ty^T \xrightarrow{f \otimes g} \texttt{Plant} \otimes \texttt{Controller}$$

In other words we have a morphism of polynomials $STy^{ST} \to \texttt{Plant} \otimes \texttt{Controller}$; that's a new dynamical system with state space $ST = (S \times T)$; a state in it is just a pair of states, one in S and one in T. Furthermore our wiring diagram already gave us a map $\texttt{Plant} \otimes \texttt{Controller} \to \texttt{System}$, so combining, we have a new system

$$STy^{ST} \to \texttt{System}.$$

Exercise 4.141. Consider the following wiring diagram.

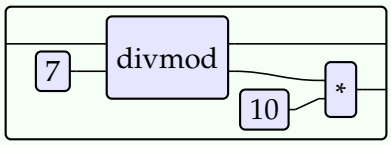


1. Write out the polynomials for each of Alice, Bob, and Carl.
2. Write out the polynomial for the outer box, Team.
3. The wiring diagram constitutes a morphism f in **Poly**; what is its type $f : ? \to ?$
4. What morphism is it?
5. Suppose we are given dynamical systems $Ay^A \to \texttt{Alice}$, $By^B \to \texttt{Bob}$, and $Cy^C \to \texttt{Carl}$. What is the induced dynamical system on `Team`? ◊

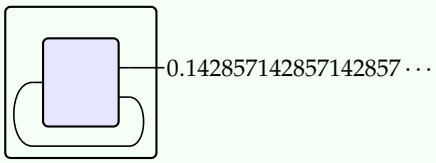
Exercise 4.142 (Long division).

1. Come up with a function "divmod" of type $\mathbb{N} \times \mathbb{N}_{\geq 1} \to \mathbb{N} \times \mathbb{N}$ and which, for example, sends $(10, 7)$ to $(1, 3)$ and $(30, 7)$ to $(4, 2)$.
2. Use Exercise 4.117 to turn it into a dynamical system.

3. Interpret the following wiring diagram:

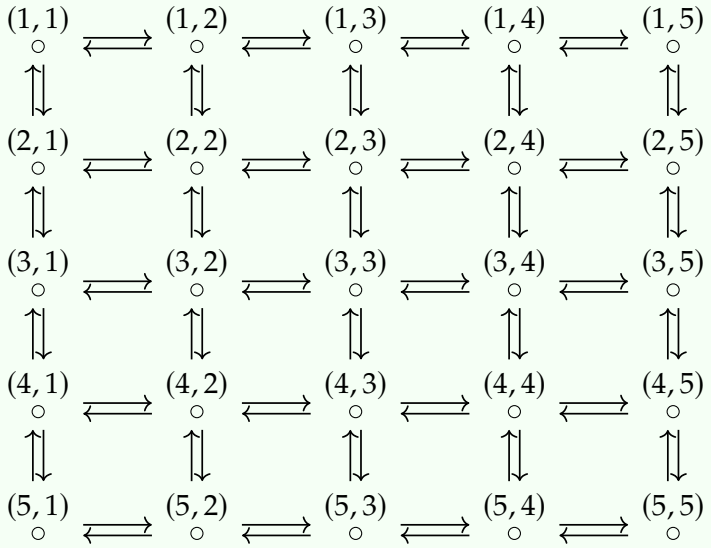


4. Use the above and a diagram of the following form to create a function that spits out the base-10 digits of $1/7$.



◊

Exercise 4.143 (Cellular automata). Let $G = (V, E)$ be a simple graph, i.e. $V, E \in \mathbf{Set}$ and $E \subseteq V \times V$. You can imagine it as a grid



finite or infinite, or just an arbitrary graph. For every vertex v, the set of vertices v' for which $(v', v) \in E$, i.e. for which there is an edge $v' \to v$, is denoted

$$I(v) \coloneqq \{v' \mid (v', v) \in E\}.$$

For each $v \in V$, let $p_v \coloneqq \mathsf{2}y^{\mathsf{2}^{I(v)}}$; it "outputs" a color $\mathsf{2} \cong \{\texttt{black}, \texttt{white}\}$ and inputs a function $I(v) \to \mathsf{2}$, specifying what all the neighbors are outputting.

1. In the drawn grid, what is $I(1, 1)$? What is $I(2, 2)$?

2. Specify a morphism $g\colon \bigotimes_{v\in V} p_v \to y$ that passes to each vertex v the colors of its neighbors in $I(v)$.
3. Suppose that for each vertex $v \in V$ you are given a function $f_v\colon 2^{I(v)} \to 2$. Use it to construct a dynamical system $f'_v\colon 2y^2 \to p_v$ that updates its state in keeping with f_v and outputs its state directly.
4. Briefly look up cellular automata in a reference of your choice. Would you say that the dynamical system $\bigotimes_{v\in V} 2y^2 \xrightarrow{\bigotimes f'_v} \bigotimes_{v\in V} p_v \xrightarrow{g} y$ we obtain by wiring together the dynamical systems in the specified way does the same thing as the cellular automata in your reference? ◊

4.3.4 General interaction

In general, we want systems that can change their interface—remove a port, add a port, change the type of a port, etc.—based on their internal states. But when such systems interact with others, the interaction pattern must be able to accommodate all of the various combinations of interfaces.

Example 4.144. Suppose given two interfaces p and p', having mode sets M and M' respectively

$$p \coloneqq \sum_{m\in M} B_m y^{A_m} \qquad \text{and} \qquad p' \coloneqq \sum_{m'\in M'} B'_m y^{A'_m}$$

The parallel product of these is:

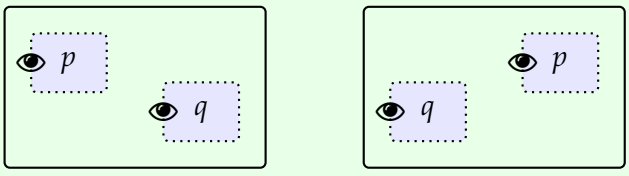
$$p \otimes p' \cong \sum_{(m,m')\in MM'} B_m B'_m y^{A_m A'_m}$$

so the interface $B_m B'_{m'} y^{A_m A'_{m'}}$ at each of these $(M \times M')$-many modes (m, m') must be accommodated in any morphism $w\colon p \otimes p' \to q$. For example if $M = 2$ and $M' = 3$ then w can be specified by six maps.

But in fact the possibilities for interaction are much more general than we have led the reader to believe. They may not be broken down into modes at all.

Example 4.145. Let $p \coloneqq By^A$ and $p' = B'y^{A'}$. To give a morphism $f\colon p \otimes p' \to y$, one specifies a map $B \times B' \to 1$, which is no data, as well as a map $BB' \to AA'$. In other words, for every pair of outputs (b, b') one specifies a pair of inputs (a, a').

Let's think of elements of B and B' not as outputs, but as positions.



Then given both positions (b, b'), the interaction pattern f tells us what the two eyes see, i.e. what values of (a, a') we get.

Example 4.146. Suppose you have two systems p, q each of type $p, q \coloneqq \mathbb{R}^2 y^{\mathbb{R}^2-(0,0)}$.

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 outer system to be of type $\{\texttt{go}\}y + \{\texttt{stop}\}$. The morphism $\mathbb{R}^2 y^{\mathbb{R}^2-(0,0)} \otimes \mathbb{R}^2 y^{\mathbb{R}^2-(0,0)} \to \{\texttt{go}\}y + \{\texttt{stop}\}$ is given on positions by

$$\big((x_1, y_1), (x_2, y_2)\big) \mapsto \begin{cases} (\texttt{stop}, 1) & \text{if } x_1 = x_2 \text{ and } y_1 = y_2 \\ (\texttt{go}, 1) & \text{otherwise.} \end{cases}$$

On directions, we use the function

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

Now each system is able to see the vector pointing from it to the other system (unless that vector is zero, in which case the whole thing has halted). Let's use these vectors to define the internal dynamics of each system. Each system will hold as its internal state its current position and velocity, i.e. $S = \mathbb{R}^2 \times \mathbb{R}^2$. To define a map of polynomials $Sy^S \to \mathbb{R}^2 y^{\mathbb{R}^2-(0,0)}$ we simply output the current position and update the current velocity by adding a vector pointing to the other system and having appropriate magnitude:

$$\mathbb{R}^2 \times \mathbb{R}^2 \xrightarrow{\text{get}} \mathbb{R}^2$$
$$\big((x, y), (x', y')\big) \xmapsto{\text{get}} (x, y)$$

$$\mathbb{R}^2 \times \mathbb{R}^2 \times (\mathbb{R}^2 - (0,0)) \xrightarrow{\text{put}} \mathbb{R}^2 \times \mathbb{R}^2$$
$$\big((x, y), (x', y'), (a, b)\big) \xmapsto{\text{put}} \left(x + x', y + y', x' + \frac{a}{(a^2 + b^2)^{3/2}}, y' + \frac{b}{(a^2 + b^2)^{3/2}}\right)$$

Exercise 4.147. Let $p, q \coloneqq \mathbb{N}y^{\mathbb{N}}$.

1. Write a polynomial morphism $p \otimes q \to y$ that corresponds to the function $(a, b) \mapsto (b, a + b)$.
2. Write dynamical systems $\mathbb{N}y^{\mathbb{N}} \to p$ and $\mathbb{N}y^{\mathbb{N}} \to q$, each of which simply outputs the previous input.
3. Suppose each system starts in state $1 \in \mathbb{N}$. What is the trajectory of the p-system? ◊

Exercise 4.148. Suppose (X, d) is a metric space, i.e. X is a set and $d\colon X \times X \to \mathbb{R}$ is a 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 $p \coloneqq AXy^{A'X}$ and $p' \coloneqq A'Xy^{AX}$, and imagine there are two robots, one with interface p and one with interface p'.

1. Write down a morphism $p \otimes p' \to y$ such that each robot receives the other's location, but that it only receives the other's signal when the locations x, x' are sufficiently close, $d(x, x') < 1$. Otherwise it receives the default signal.
2. Write down a morphism $p \otimes p' \to y^{[0,5]}$ where the value $s \in [0, 5]$ is a scalar, allowing the signal to travel s times further.
3. Suppose that each robot has a set S, S' of private states. What functions are involved in providing a dynamical system $f\colon SXy^{SX} \to AXy^{A'X}$?
4. Change the setup in any way so that the robots only extend a port to hear the other's signal when the distance between them is less than 1. Otherwise, they can only detect the position (element of X) that the other currently inhabits. ◊

So what is a map $p_1 \otimes \cdots \otimes p_k \to q$ in general? It's a protocol by which he k-many participants p_i together decide what decision q must make, as well as how q's choice among its options (the decision once made) is passed back and distributed as an option at each p_i.

Example 4.149 (Cellular automata who vote on their interaction pattern). Recall from Exercise 4.143 how we constructed cellular automata on a graph $G = (V, E)$. Here $E \subseteq V \times V$, or equivalently what we might call an *interaction pattern* $I\colon V \to 2^V$, specifies the incoming neighbors $I(v)$ of each $v \in V$.

Suppose now that we are given a function $i\colon V \to \mathbb{N}$ that we think of as specifying the number $i(v)$ of neighbors each $v \in V$ accepts. Let $\mathsf{i}(v) = \{1, 2, \ldots, i(v)\}$. We will be interested in the polynomial $p_v \coloneqq 2y^{2^{\mathsf{i}(v)}}$ for each v; it represents an interface that outputs a color $2 \cong \{\texttt{black}, \texttt{white}\}$ and that inputs a function $\mathsf{i}(v) \to 2$, meant to give the colors of the neighboring vertices.

Say that an interaction pattern $I\colon V \to 2^V$ *respects* i if we have an isomorphism $I(v) \cong \mathsf{i}(v)$ for each $v \in V$. Suppose given a function $I\colon 2^V \to (2^V)^V$ such that for

each element $s \in 2^V$, the interaction pattern $I_s \colon V \to 2^V$ respects i. In the case of Exercise 4.143, I was a constant function. Now we can think of it like all the vertices are voting, via I, on the connection pattern.

We can put this all together by giving a morphism in **Poly** of the form

$$\bigotimes_{v \in V} p_v \cong 2^V y^{2^{\sum_{v \in V} \mathsf{i}(v)}} \longrightarrow y. \tag{4.150}$$

We can such a morphism with a function $g \colon 2^V \to 2^{\sum_{v \in V} \mathsf{i}(v)}$. Suppose given $s \in 2^V$, so that we have an isomorphism $I_s(v) \cong \mathsf{i}(v)$ for each $v \in V$; we want a function $g(s) \colon \sum_{v \in V} I_s(v) \to 2$. That is, for each v we want a function

$$g(s)_v \colon I_s(v) \to 2.$$

But $I_s(v) \subseteq V$, so our function $s \colon V \to 2$ induces the desired function $I_s(v) \to 2$.

We have accomplished our goal: the automata vote on their connection pattern. 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 $I \colon 2^V \to (2^V)^V$. It could be dictated by a given vertex $v_0 \in V$ in the sense that its on/off state completely determines the connection pattern $V \times V \to 2$; this would be expressed by saying that I factors as $2^V \to 2^{v_0} \cong 2 \xrightarrow{I_0} (2^V)^V$ for some I_0.

Exercise 4.151. Change Example 4.149 slightly by changing the outer box.

1. First change it to Ay for some set A of your choice, and update (4.150) so that the system outputs some aspect of the current state 2^V.
2. What would it mean to change (4.150) to a map $\bigotimes_{v \in V} p_v \to y^A$ for some A? ◊

Example 4.152. Recall the picture from Example 4.8. 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 systems Φ_1 and Φ_2 as initially connected in space, that they experience forces from the outside world, and that—for as long as they are connected—they experience forces from each other. More precisely, each internal arena is defined by

$$p_1 = p_2 \coloneqq F y^{FF} + y^F.$$

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 arena has two modes: the monomial Fy^{FF} consisting of two input forces (one from its left and one from its right) and an output force f_i, and the monomial y^F consisting of one input force (just from the outside). Similarly, in the first mode the system Φ_i is outputting a force for the other—whether the other uses it or not—but in the second mode the system produces no force for the other.

The external arena is defined to be

$$p \coloneqq y^{FF};$$

it takes as input two forces (f_L, f_R) and produces unchanging output.

Though the systems Φ_1 and Φ_2 may be initially connected, if the forces on either one surpass a threshold, that system stops sending and receiving forces from the other. The connection is broken and neither system ever receives forces from the other again. This is what we will implement explicitly below.

To do so, we need to create a contract $p_1 \otimes p_2 \to p$ of the external arena p around (the arenas of) the internal systems. That is, we need to give a morphism of polynomials

$$\kappa \colon (Fy^{FF} + y^F) \otimes (Fy^{FF} + y^F) \to y^{FF}.$$

By distributivity and the universal property of coproducts, it suffices to give four maps:

$$\begin{aligned} \kappa_{11} &\colon FFy^{(FF)(FF)} \to y^{FF} \\ \kappa_{12} &\colon Fy^{(FF)F} \to y^{FF} \\ \kappa_{21} &\colon Fy^{F(FF)} \to y^{FF} \\ \kappa_{22} &\colon y^{FF} \to y^{FF} \end{aligned}$$

The middle two maps κ_{12} and κ_{21} won't actually occur in our dynamics, so we take them to be arbitrary. We take the last map κ_{22} to be an identity (the forces from outside are passed to the two internal boxes). The first map κ_{11} is equivalent to a function $(FF)(FF) \to (FF)(FF)$ which we take to be $((f_1, f_2), (f_L, f_R)) \mapsto ((f_L, f_2), (f_1, f_R))$.

Now that we have the arenas wired together, it remains to give the dynamics on the internal boxes. The states in the two cases will be identical, namely $S \coloneqq F + 1$, meaning that at any point the system will either be in the state of holding a force or not. The dynamics will be identical as well, up to a symmetry swapping left and right; let's work with the first. Its interface is $p_1 = Fy^{FF} + y^F$ and its dynamics are given by

$$\Phi_1 \colon (F+1)y^{F+1} \to Fy^{FF} + y^F$$

which splits up as the coproduct of $Fy^{F+1} \to Fy^{FF}$ and $y^{F+1} \to y^F$. The second map corresponds to when the connection is broken; it is given by projection, meaning it just updates the state to be the received force. The first map $Fy^{F+1} \to Fy^{FF}$ corresponds to the case where the system is holding some force, is receiving two input forces and must update its state and produce one output force. For the passforward $F \to F$,

let's use identity meaning it outputs the force it's holding. For the passback $F(FF) \to \{\texttt{Just}\}F + \{\texttt{Nothing}\}$, let's use the map $(f, (f_L, f_2)) \mapsto t(f_L, f_2)$ defined here:

$$t(f_L, f_2) := \begin{cases} \texttt{Just } f_L & \text{if } f_1 + f_2 < 100 \\ \texttt{Nothing} & \text{otherwise} \end{cases}$$

Thus when the sum of forces is high enough, the internal state is updated to the broken state; otherwise it is sent to the force it receives from outside.

Example 4.153. We want to consider the case of a company C that may change its supplier based on its internal state. The company has no output wires, but has two modes of operation—two positions—corresponding to who it wants to receive widgets W from:

The company has interface $2y^W$, and the each supplier has interface Wy; let's take the total system interface (undrawn) to be the closed system y. Then this mode-dependent wiring diagram is just a map $2y^W \otimes Wy \otimes Wy \to y$. Its on-positions function $2W^2 \to 1$ is uniquely determined, and its on-directions function $2W^2 \to W$ is the evaluation. In other words, the company's position determines which supplier from which it receives widgets.

Example 4.154. When someone assembles a machine, their own outputs dictate the connection pattern of the machine's components.

(4.155)

In order for the above picture to make sense, A has the same output that B has as input, say X, and we need a default value $x_0 \in X$ for B to input when not connected to A.

We could say that the person in (4.155) has interface $2y$, the units have interfaces Xy and y^X respectively, and the whole system is closed; that is, the diagram represents a morphism $2y \otimes Xy \otimes y^X \to y$. The morphism $2Xy^X \to y$ is uniquely determined on positions, and on directions it is given by cases $(1, x) \mapsto x_0$ and $(2, x) \mapsto x$.

We can easily generalize Example 4.154. Indeed, we will see in the next section that there is a polynomial $[(]q_1 \otimes \cdots \otimes q_k\,,\, r)$ of all ways $q_1, \ldots, q_k$ can interact in r, and that a map from some p to it is just a bigger 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 thinks its deciding how $q_1, \ldots, q_k$ are wired up in r, and gets feedback from that wiring pattern itself, then in actuality p is just part of a wiring diagram with $q_1, \ldots, q_k$ inside of r.

What it also means is that if you want, you can put a little dynamical system inside of $[q_1 \otimes \cdots \otimes q_k, r]$ and have it constantly choosing interaction patterns. Let's see how it works.

4.3.5 Closure of $\otimes$

The parallel monoidal product is closed, meaning that there is an operation, which we denote $[-,-]\colon \mathbf{Poly}^{\mathrm{op}} \times \mathbf{Poly} \to \mathbf{Poly}$ and an isomorphism

$$\mathbf{Poly}(p \otimes q, r) \cong \mathbf{Poly}(p, [q, r])$$

natural in p, q, r. The closure operation is defined on q, r as follows:

$$[q, r] \coloneqq \prod_{j \in q(1)} r \circ (q[j]y) \tag{4.156}$$

Example 4.157. For any $A \in \mathbf{Set}$ we have

$$[y^A, y] \cong Ay \qquad \text{and} \qquad [Ay, y] \cong y^A.$$

These both follow directly from (4.156).

Exercise 4.158. Calculate $[q, r]$ for $q, r \in \mathbf{Poly}$ given as follows.

1. $q \coloneqq 0$ and r arbitrary.
2. $q \coloneqq 1$ and r arbitrary.
3. $q \coloneqq y$ and r arbitrary.
4. $q \coloneqq A$ and $r \coloneqq B$ for $A, B \in \mathbf{Set}$ (constants).
5. $q \coloneqq Ay$ and $r \coloneqq By$ for $A, B \in \mathbf{Set}$ (linears).
6. $q \coloneqq 2y^3 + y^2$ and $r \coloneqq 4y^2 + 7$. ◊

Exercise 4.159. Show that for any $p \in \mathbf{Poly}$, if there is an isomorphism $[[p, y], y] \cong p$, then p is either linear Ay or representable y^A for some A. Hint: first show that p must be a monomial. ◊

Proposition 4.160. With $[-,-]$ as defined in (4.156), there is a natural isomorphism

$$\mathbf{Poly}(p \otimes q, r) \cong \mathbf{Poly}(p, [q, r]). \tag{4.161}$$

Proof. We have the following chain of natural isomorphisms:

$$\begin{aligned}
\mathbf{Poly}(p \otimes q, r) &\cong \mathbf{Poly}\Big(\sum_{i \in p(1)} \sum_{j \in q(1)} y^{p[i]q[j]}, r\Big) \\
&\cong \prod_{i \in p(1)} \prod_{j \in q(1)} \mathbf{Poly}(y^{p[i]q[j]}, r) \\
&\cong \prod_{i \in p(1)} \prod_{j \in q(1)} r(p[i]q[j]) \\
&\cong \prod_{i \in p(1)} \prod_{j \in q(1)} \mathbf{Poly}(y^{p[i]}, r \circ (q[j]y)) \\
&\cong \prod_{i \in p(1)} \mathbf{Poly}\Big(y^{p[i]}, \prod_{j \in q(1)} r \circ (q[j]y)\Big) \\
&\cong \mathbf{Poly}(p, [q, r]).
\end{aligned}$$

□

Exercise 4.162. Show that for any p, q we have an isomorphism of sets

$$\mathbf{Poly}(p, q) \cong [p, q](1).$$

Hint: you can either use the formula (4.156), or just use (4.161) with the Yoneda lemma and the fact that $y \otimes p \cong p$. ◊

For any $p, q \in \mathbf{Poly}$ there is a canonical *evaluation* map

$$\mathrm{eval} \colon [p, q] \otimes p \longrightarrow q. \tag{4.163}$$

Exercise 4.164.

1. Obtain the evaluation map $\mathrm{eval} \colon [p, q] \otimes p \longrightarrow q$ from (4.161).
2. Show that for any p, q, r and map $f \colon p \otimes q \to r$, there is a unique morphism $f' \colon p \to [q, r]$ such that the following diagram commutes

$$p \otimes q \xrightarrow{f' \otimes q} [q, r] \otimes q \xrightarrow{\mathrm{eval}} r, \qquad p \otimes q \xrightarrow{f} r$$

◊

Exercise 4.165.

1. For any S, obtain the map $Sy^S \to y$ whose on-directions map is identity, using eval and Example 4.157.
2. Show that maps of the four types κ_{11}, κ_{12}, κ_{21}, κ_{22} shown in Example 4.152 can be obtained by tensoring together identity maps and eval maps. ◊

Example 4.166 (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 living together in a closed system

$$f : (q_1 \otimes \cdots \otimes q_n) \otimes p \to y$$

with some other robots whose interfaces are $q_1, \ldots, q_n$; let $q \coloneqq (q_1 \otimes \cdots \otimes q_n)$. The interaction pattern induces a morphism $f' : q \to [p, y]$ such that the original system f factors through the evaluation $[p, y] \otimes p \to y$.

In other words $[p, y]$ holds within it all of the possible ways p can interact with other systems in a closed box.[a] To investigate this just a bit, note that $[p, y] \cong \prod_{i \in p(1)} p[i]y$. That is, for each position in p it produces a direction there, which is just what p needs as input.

Now suppose we were to populate the interface p with dynamics, a map $Sy^S \to p$. One could aim to choose a set S along with an interesting map $g : S \to \mathbf{Poly}(p, y)$. Then each state s would include a guess $g(s)$ about what the state of the environment is in. This is not the real environment q, but just the environment as it affects p, namely $[p, y]$. The robot's states model environmental conditions.

[a] And if you want the generic way p to interact with other systems in a box r, just use $[p, r]$.

4.4 Bonus math about the category Poly

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 Chapter 5 if so inclined. Better yet might be to just gently leaf through Section 4.4, to see how well-behaved and versatile the category **Poly** really is.

4.4.1 Special polynomials and adjunctions

There are a few special classes of polynomials that are worth discussing:

1. constant polynomials $0, 1, 2, A$;
2. linear polynomials $0, y, 2y, Ay$;
3. pure-powers polynomials, $1, y, y^2, y^A$; and
4. monomials $0, A, y, 2y^3, Ay^B$.

The first two classes, constant and linear polynomials, are interesting because they both put a copy of **Set** inside **Poly**, as we'll see in Propositions 4.168 and 4.169. The third puts a copy of $\mathbf{Set}^{\mathrm{op}}$ inside **Poly**, and the fourth puts a copy of bimorphic lenses inside **Poly**, as we saw in Section 4.2.4 (page 196).

Exercise 4.167. Which of the four classes above are closed under

1. the cocartesian monoidal structure $(0, +)$?
2. the cartesian monoidal structure $(1, \times)$?
3. the parallel monoidal structure $(y, \otimes)$?
4. composition of polynomials $p \circ q$?[a] ◊

[a] Composition, together with the unit y, is in fact yet another monoidal structure, as we will see in Chapter 5.

Proposition 4.168. There is a fully faithful functor **Set** $\to$ **Poly** sending $A \mapsto Ay^0 = A$.

Proof. By Eq. (4.59), a map $f : Ay^0 \to By^0$ consists of a function $f : A \to B$ and for each $a \in A$ a function $0 \to 0$. There is only one such function, so f can be identified with just a map of sets $A \to B$. □

Proposition 4.169. There is a fully faithful functor **Set** $\to$ **Poly** sending $A \mapsto Ay$.

Proof. By Eq. (4.59), a map $f : Ay^1 \to By^1$ consists of a function $f : A \to B$ and for each $a \in A$ a function $1 \to 1$. There is only one such function, so f can be identified with just a map of sets $A \to B$. □

Theorem 4.170. **Poly** has an adjoint quadruple with **Set**:

$$\mathbf{Set} \;\; \underset{\overset{\overset{p(0)}{\longleftarrow}}{\Rightarrow}}{\overset{A}{\longrightarrow}} \;\; \underset{\overset{\overset{p(1)}{\longleftarrow}}{\Leftarrow}}{\overset{Ay}{\longrightarrow}} \;\; \mathbf{Poly} \tag{4.171}$$

where the functors have been labeled by where they send $A \in \mathbf{Set}$ and $p \in \mathbf{Poly}$.

Both rightward functors are fully faithful.

Proof. For any set A, there is a functor **Poly** $\to$ **Set** given by sending p to $p(A)$; it is $\mathbf{Poly}(A, -)$. This, together with Propositions 4.168 and 4.169 give us the four functors and the fact that the two rightward functors are fully faithful. It remains to provide the following three natural isomorphisms:

$$\mathbf{Set}(A, p(0)) \cong \mathbf{Poly}(A, p) \qquad \mathbf{Set}(p(1), A) \cong \mathbf{Poly}(p, A) \qquad \mathbf{Set}(A, p(1)) \cong \mathbf{Poly}(Ay, p).$$

All three come from our main formula, Eq. (4.63); we leave the details to the reader in Exercise 4.172. □

Exercise 4.172. Here we prove the remainder of Theorem 4.170 using Eq. (4.63):

1. Provide a natural isomorphism $\mathbf{Set}(A, p(0)) \cong \mathbf{Poly}(A, p)$.
2. Provide a natural isomorphism $\mathbf{Set}(p(1), A) \cong \mathbf{Poly}(p, A)$.
3. Provide a natural isomorphism $\mathbf{Set}(A, p(1)) \cong \mathbf{Poly}(Ay, p)$. ◊

Exercise 4.173. Show that for any polynomial p, its set $p(1)$ of positions is in bijection with the set of functions $y \to p$. ◊

In Theorem 4.170 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 4.176. However, the fact that $p \mapsto p(1)$ is also a right adjoint—and hence that we have the *quadruple* of adjunctions in (4.171)—is special to $A = 0, 1$.

Next we note that the set of polynomial morphisms $p \to q$

Proposition 4.174. There is a two-variable adjunction between **Set**, **Poly**, and **Poly**:[a]

$$\mathbf{Poly}(Ap, q) \cong \mathbf{Poly}(p, q^A) \cong \mathbf{Set}(A, \mathbf{Poly}(p, q)). \tag{4.175}$$

[a] The first set $\mathbf{Poly}(Ap, q)$ involves the A-fold coproduct of p and the middle set $\mathbf{Poly}(p, q^A)$ involves the A-fold product of q, neither of which we have proven exists. For organizational purposes, we put that off until Proposition 4.187.

Proof. Since Ap is the A-fold coproduct of p and q^A is the A-fold product of q, the universal properties of coproduct and product give isomorphisms

$$\mathbf{Poly}(Ap, q) \cong \prod_{a \in A} \mathbf{Poly}(p, q) \cong \mathbf{Poly}(p, q^A).$$

The middle set is in bijection with $\mathbf{Set}(A, \mathbf{Poly}(p, q))$, completing the proof. □

Replacing q with p and replacing p with y^B in Proposition 4.174, we obtain the following using the Yoneda lemma.

Corollary 4.176. For any set B there is an adjunction

$$\mathbf{Set} \underset{p(B)}{\overset{Ay^B}{\underset{\longleftarrow}{\overset{\longrightarrow}{\Rightarrow}}}} \mathbf{Poly}$$

where the functors are labeled by where they send $p \in \mathbf{Poly}$ and $A \in \mathbf{Set}$.

Exercise 4.177. Prove Corollary 4.176 from Proposition 4.174. ◊

Proposition 4.178. The Yoneda embedding $A \mapsto y^A$ has a left adjoint

$$\mathbf{Set}^{\mathrm{op}} \underset{\Gamma}{\overset{y^-}{\underset{\longleftarrow}{\overset{\longrightarrow}{\Leftarrow}}}} \mathbf{Poly}$$

where $\Gamma(p) \coloneqq \prod_{i \in p(1)} p[i]$.

Proof. We have the following chain of natural isomorphisms:

$$\begin{aligned}
\mathbf{Set}(A, \Gamma(p)) &\cong \Big(\prod_{i \in p(1)} p[i] \Big)^A && \text{Definition of function} \\
&\cong \prod_{i \in p(1)} p[i]^A && \text{Definition of product} \\
&\cong \prod_{i \in p(1)} \sum_{j \in 1} p[i]^A && \text{Trivial sum} \\
&\cong \mathbf{Poly}(p, y^A). && Eq.\ (4.63)
\end{aligned}$$

□

Exercise 4.179. Show that $\Gamma(p) \cong [p, y](1)$ where $[-,-]$ is as in Proposition 4.160. ◊

Epi-mono factorization.

Proposition 4.180. Let $(f_1, f^\sharp) \colon p \to q$ be a morphism in **Poly**. It is a monomorphism iff the function $f_1 \colon p(1) \to q(1)$ is a monomorphism in **Set** and, for each $i \in p(1)$ the function $f_i^\sharp \colon q[f_1(i)] \to p[i]$ is an epimorphism in **Set**.

Proof. $\Rightarrow$: Suppose that f is a monomorphism. Since $p \mapsto p(1)$ is a right adjoint (Theorem 4.170), it preserves monomorphisms. We need to show that for any $i \in p(1)$ the function $f_i^\sharp \colon q[f_1(i)] \to p[i]$ is an epimorphism in **Set**. Suppose given a set A and a pair of functions $g^\sharp, h^\sharp \colon p[i] \rightrightarrows A$ with $g^\sharp f_i^\sharp = h^\sharp f_i^\sharp$. They can be identified with morphisms $y^A \rightrightarrows p$ whose compositions with f are equal, hence $g = h$ by assumption, and hence $g^\sharp = h^\sharp$ as desired.
$\Leftarrow$: 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, $g, h \colon r \rightrightarrows p$ two morphisms, and suppose $fg = fh$. Then $f_1 g_1 = f_1 h_1$, which implies $g_1 = h_1$; we'll consider g_1 the default representation. We also have that $g_k^\sharp f_{g_1 k}^\sharp = h_k^\sharp f_{g_1 k}^\sharp$ for any $k \in r(1)$. But $f_{g_1 k}^\sharp$ is an epimorphism, so in fact $g_k^\sharp = h_k^\sharp$, as desired. □

Example 4.181. Choose a finite nonempty set k for $1 \leq k \in \mathbb{N}$, e.g. $\mathsf{k} = \mathsf{12}$. There is a monomorphism

$$(f, f^\sharp) \colon \mathsf{k}y^{\mathsf{k}} \to \mathbb{N}y^{\mathbb{N}}$$

such that the trajectory "going around and around the k-clock" comes from the usual counting trajectory Example 4.113 $\mathbb{N}y^{\mathbb{N}} \to y$.

On positions we have $f(i) = i$, for all natural numbers $1 \leq i \leq k$. On directions we have $f^\sharp(n) = n \mod k$.

Exercise 4.182. In Example 4.181 we gave a map $\mathsf{12}y^{\mathsf{12}} \to \mathbb{N}y^{\mathbb{N}}$. This allows us to turn any dynamical system with $\mathbb{N}$-many states into a dynamical system with 12 states, while keeping the same interface, say p.

Explain how the behavior of the new system $\mathsf{12}y^{\mathsf{12}} \to p$ would be seen to relate to the behavior of the old system $\mathbb{N}y^{\mathbb{N}} \to p$. ◊

Proposition 4.183. Let $(f_1, f^\sharp) \colon p \to q$ be a morphism in **Poly**. It is an epimorphism iff the function $f_1 \colon p(1) \to q(1)$ is an epimorphism in **Set** and, for each $j \in q(1)$ the induced function

$$f_j^\flat \colon q[j] \to \prod_{\{i \in p(1) \mid f_1(i) = j\}} p[i]$$

from (4.79) is a monomorphism.

Proof. $\Rightarrow$: Suppose that f is an epimorphism. Since $p \mapsto p(1)$ is a left adjoint (Theorem 4.170), it preserves epimorphisms. We need to show that for any $j \in q(1)$ the function $f_j^\flat$ is a monomorphism in **Set**. Suppose given a set A and a pair of functions $g^\sharp, h^\sharp \colon A \rightrightarrows q[j]$ with $f_j^\flat g^\sharp = f_j^\flat h^\sharp$. They can be identified with morphisms $g, h \colon q \rightrightarrows y^A + 1$, which send the j-component to the first component, y^A, and send all other component to the second component, 1. It is easy to check that $fg = fh$, hence $g = h$, and hence $g^\sharp = h^\sharp$ as desired.

$\Leftarrow$: Suppose that f_1 is an epimorphism and that for each $j \in q(1)$ the function $f_j^\flat$ is a monomorphism. Let r be a polynomial, $g, h \colon q \rightrightarrows r$ two morphisms, and suppose $gf = hf$. Then $g_1 f_1 = h_1 f_1$, which implies $g_1 = h_1$; we'll consider g_1 the default representation. We also have that $f_i^\sharp g^\sharp_{f i} = f_i^\sharp h^\sharp_{f i}$ for any $i \in p(1)$. Then, in particular, for any $j \in q(1)$ the two composites

$$r[g_1 j] \underset{h_j^\sharp}{\overset{g_j^\sharp}{\rightrightarrows}} q[j] \xrightarrow{f_j^\flat} \prod_{\{i \in p(1) \mid f_1(i) = j\}} p[i]$$

are equal, which implies that $g_j^\sharp = h_j^\sharp$ as desired. □

Exercise 4.184. Suppose f is both a mono and an epi; it is an iso? (That is, is **Poly** *balanced*?) ◊

Exercise 4.185 (Epi-mono factorization). Suppose $p, q \in \mathbf{Poly}$ are polynomials and $f = (f_1, f^\sharp) \colon p \to q$ is a morphism with notation as in Eq. (4.59).

1. Can every morphism in **Poly** be factored as an epic followed by a monic?
2. Is your factorization unique up to isomorphism? ◊

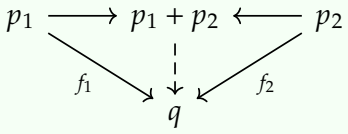
4.4.2 Limits, colimits, and cartesian closure

We will now see that **Poly** has all limits and colimits, and moreover it is cartesian closed.

Let's begin with something fairly simple: that + gives a coproduct in **Poly**. We could have said this elementary fact much earlier, but we were in a rush to get to the applications in Section 4.3.

Exercise 4.186. Show that the category **Poly** has finite coproducts as follows.

1. Show that 0 is an initial object in **Poly**, i.e. that for any polynomial p there is a unique morphism $0 \to p$.
2. Show that the standard sum of polynomials $p_1 + p_2$ is a coproduct of p_1 and p_2. That is, provide morphisms $p_1 \to p_1 + p_2 \leftarrow p_2$ and show that for any other q with morphisms $p_1 \xrightarrow{f_1} q \xleftarrow{f_2} p_2$, there exists a unique morphism $p_1 + p_2 \to q$—shown dashed—making the following diagram commute



◊

The general case of coproducts and products is not much more difficult.

Proposition 4.187. The category **Poly** has coproducts and products.

Proof. Let A be a set and $p \colon A \to \mathbf{Poly}$ a collection of polynomials $(p_a)_{a \in A}$.Their coproduct is

$$\sum_{a \in A} p_a = \sum_{a \in A} \sum_{i \in p_a(1)} y^{p_a[i]} \cong \sum_{(a,i) \in \sum_{a \in A} p_a(1)} y^{p_a[i]} \tag{4.188}$$

which is manifestly a sum of representables, i.e. a polynomial, and which one can check satisfies the universal property of coproduct; see Exercise 4.190.

We claim that the product of this collection is

$$\prod_{a \in A} p_a = \prod_{a \in A} \sum_{i \in p_a(1)} y^{p_a[i]} \cong \sum_{i \colon \prod_{a \in A} p_a(1)} y^{\sum_{a \in A} p_a[i(a)]}. \tag{4.189}$$

Again, it is a sum of representables, so a polynomial, and one can check that it satisfies the universal property of product; see Exercise 4.190. □

Exercise 4.190. Finish the proof of Proposition 4.187 as follows.

1. Show that (4.188) satisfies the universal property for coproduct of polynomials.
2. Show that (4.189) satisfies the universal property for product of polynomials. ◊

Exercise 4.191. Let $A = 2$ and $p\colon A \to \mathbf{Poly}$ be given by $p_1 := y + 1$ and $p_2 := y + 2$. What is $\prod_{i\in 2} p[i]$ according to Eq. (4.189)? Does it check out? ◊

Proposition 4.192. The category **Poly** is completely distributive, i.e. for any set A, sets $(B_a)_{a\in A}$, and polynomials $(p_{(a,b),})_{a\in A, b\in B_a}$, there is an isomorphism

$$\sum_{b\colon \prod_{a\in A} B(a)} \prod_{a\in A} p_{(a,b(a)),} \cong \prod_{a\in A} \sum_{b\in B(a)} p_{(a,b),}$$

Proof. By the universal property of coproducts and products, to provide a morphism $\sum_{b\colon \prod_{a\in A} B(a)} \prod_{a\in A} p_{(a,b),} \to \prod_{a\in A} \sum_{b\in B(a)} p_{(a,b),}$, it suffices to fix an arbitrary $b_0 \in \prod_{a\in A} B(a)$ and $a_0 \in A$ and provide a morphism $\prod_{a\in A} p_{(a,b_0(a)),} \to \sum_{b\in B(a_0)} p_{(a_0,b),}$. We do so by first projecting onto the a_0-factor $\prod_{a\in A} p_{(a,b_0(a)),} \to p_{(a_0,b_0(a_0)),}$ and then including into the $b_0(a_0)$-summand $p_{(a_0,b_0(a_0)),} \to \sum_{b\in B(a_0)} p_{(a_0,b),}$. This establishes the morphism.

To see that it is an isomorphism in **Poly**, i.e. a natural isomorphism of functors, it suffices to check it on components. So fix $X \in \mathbf{Set}$. The induced map

$$\sum_{b\colon \prod_{a\in A} B(a)} \prod_{a\in A} p_{(a,b(a)),} \cong \prod_{a\in A} \sum_{b\in B(a)} p_{(a,b),}$$

is exactly as in Proposition 4.74, which we proved is an isomorphism. This completes the proof. □

Exercise 4.193. How is the usual distributive law,

$$p(q + r) \cong pq + pr$$

for $p, q, r \in \mathbf{Poly}$, a special case of Proposition 4.192? ◊

Cartesian closure For any two polynomials q, r, define $r^q \in \mathbf{Poly}$ by the following formula

$$r^q := \prod_{i\in q(1)} r \circ (y + q[j]) \tag{4.194}$$

where $\circ$ denotes composition.

Before proving that this really is an exponential in **Poly**, which we do in Theorem 4.197, we first get some practice with it.

Example 4.195. Let A be a set. We've been writing the polynomial Ay^0 simply as A and the polynomial y^1 simply as y, so it better be true that the there is an isomorphism

$$y^A \cong (y^1)^{(Ay^0)}$$

in order for the notation to be consistent. Luckily, this is true. Checking (4.194), we have

$$(y^1)^{Ay^0} = \prod_{a \in A} y \circ (0 + y) \cong y^A$$

Exercise 4.196. Compute the following exponentials in **Poly** using (4.194):

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 4.197. 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 (4.194).

Proof. We have the following chain of natural isomorphisms

$$\begin{aligned}
\mathbf{Poly}(p, r^q) &\cong \mathbf{Poly}\Big(p, \prod_{j \in q(1)} r \circ (y + q[j])\Big) \\
&\cong \prod_{j \in q(1)} \mathbf{Poly}(p, r \circ (y + q[j])) \\
&\cong \prod_{j \in q(1)} \prod_{i \in p(1)} \mathbf{Poly}(y^{p[i]}, r \circ (y + q[j])) \\
&\cong \prod_{i \in p(1)} \prod_{j \in q(1)} r \circ (p[i] + q[j]) \\
&\cong \prod_{i \in p(1)} \prod_{j \in q(1)} \sum_{k \in r(1)} (p[i] + q[j])^{r[k]} \\
&\cong \mathbf{Poly}(p \times q, r).
\end{aligned}$$

The last step uses Eq. (4.63) and Proposition 4.83. □

Exercise 4.198. Use Theorem 4.197 to show that for any polynomials p, q, there is a canonical evaluation map

$$\text{eval}\colon q \times p^q \to p. \tag{4.199}$$

◊

Exercise 4.200. Using Eq. (4.194), show that the functor **Set** $\to$ **Poly** sends exponentials to exponentials. ◊

Theorem 4.201. The category **Poly** has all limits.

Proof. Recall that a category has all limits iff it has equalizers and products, so by Proposition 4.187 it suffices to show that **Poly** has equalizers.

Let $f_1, f_2\colon p \rightrightarrows q$ be two maps of polynomials. Let $E \to p(1) \rightrightarrows q(1)$ be an equalizer of the functions $f_1(1)$, and $f_2(1)$ in **Set**. For each $e \in E$, write $f(e) \coloneqq f_1(e) = f_2(e)$; we have two polynomial morphisms $y^{p_e} \rightrightarrows y_{q[f(e)]}$, i.e. two functions $q[f(e)] \rightrightarrows p_e$. Defining $p'[e] \in$ **Set** to be their coequalizer, we can define a polynomial p' as follows:

$$p' \coloneqq \sum_{e \in E} y^{p'[e]}$$

which comes equipped with a morphism $g\colon p' \to p$. One can check that it is an equalizer of f_1, f_2; see Exercise 4.202. □

Exercise 4.202. Complete the proof of Theorem 4.201 as follows:

1. We said that p' comes equipped with a morphism $g\colon p' \to p$; what is it?
2. Show that $g \fatsemi f_1 = g \fatsemi f_2$.
3. Show that g is an equalizer of the pair f_1, f_2. ◊

Exercise 4.203. Let p be any polynomial.

1. There is a canonical choice of morphism $\eta\colon p \to p(1)$; what is it?
2. Suppose given an element $i \in p(1)$, i.e. a function $1 \to p(1)$.
3. What is the pullback

$$\begin{array}{ccc} ? & \longrightarrow & p \\ \downarrow & \lrcorner & \downarrow \eta \\ 1 & \xrightarrow[i]{} & p(1) \end{array}$$

◊

Example 4.204 (Pullbacks in **Poly**). Given polynomials q, q', r and dependent lenses $q \xrightarrow{f} r \xleftarrow{f'} q'$, the pullback

$$\begin{array}{ccc} p & \longrightarrow & q' \\ \downarrow & \lrcorner & \downarrow{\scriptstyle f'} \\ q & \xrightarrow[f]{} & r \end{array}$$

is given as follows. The set of positions in p is the pullback of the positions of q and q' over those of r. On each p-position, say (i, i') with $f_1(i) = f'_1(i')$, we take the directions set of p in that position to be the pushout of the directions $q[i]$ and $q'[i']$ over $r[f_1(i)] = r[f'_1(i')]$:

$$\begin{array}{ccc} p(1) & \longrightarrow & q'(1) \\ \downarrow & \lrcorner & \downarrow{\scriptstyle f'_1} \\ q(1) & \xrightarrow[f_1]{} & r(1) \end{array} \qquad \text{and} \qquad \begin{array}{ccc} p[(i,i')] & \longleftarrow & q'[i'] \\ \uparrow & \ulcorner & \uparrow{\scriptstyle (f')^\sharp_{i'}} \\ q[i] & \xleftarrow[f_i^\sharp]{} & r[f_1(i)] \end{array} \tag{4.205}$$

Exercise 4.206. Let $q := y^2 + y$, $q' := 2y^3 + y^2$, and $r := y + 1$.

1. Choose morphisms $f : q \to r$ and $f' : q' \to r$ and write them down.
2. Find the pullback of your diagram. ◊

Arbitrary colimits will be much less useful to us than limits, so the following theorem is included only for completeness; the reader can feel free to skip it.

Theorem 4.207. The category **Poly** has all colimits.

Proof. Recall that a category has all colimits iff it has coequalizers and coproducts, so by Proposition 4.187 it suffices to show that **Poly** has coequalizers.

Let $f_1, f_2 : p \rightrightarrows q$ be two maps of polynomials. The pair of functions

$$f_1(1), f_2(1) : p(1) \rightrightarrows q(1)$$

define a graph $G : \boxed{\bullet \rightrightarrows \bullet} \to \mathbf{Set}$ whose set C of connected components is given by the coequalizer $g : q(1) \to C$ of $f_1(1)$ and $f_2(1)$. The coequalizer of f_1 and f_2 will turn out to be a C-indexed sum of representables, each of which is given by a limit of a diagram of representables from 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 $V_c := g^{-1}(c)$ and edges $E_c := f_1^{-1}(g^{-1}(c)) = f_2^{-1}(g^{-1}(c))$. 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 representable $y^{p[e]}$ (resp. $y^{q[v]}$).

The category of elements $\int G_c$ is a bipartite graph with objects $V + E$ and with two sorts of morphisms, $e \to f_1(e)$ and $e \to f_2(e)$, associated to each $e \in E_c$; all non-identity arrows point from an object in E to an object in V. There is a functor $F : (\int G_c)^{\mathrm{op}} \to \mathbf{Set}$ sending every $v \mapsto q[v]$, every $e \mapsto p[e]$, and every morphism to a function between them, namely either $(f_1^\sharp)_e : q[f_1(e)] \to p[e]$ or $(f_2^\sharp)_e : q[f_2(e)] \to p[e]$. Define $q'[c] \in \mathbf{Set}$ to be the limit $q'[c] := \lim F \in \mathbf{Set}$ of F.

We claim that $q' := \sum_{c \in C} y^{q'[c]}$ is the coequalizer of f_1 and f_2. We leave the completion proof to the interested reader in Exercise 4.208. □

Exercise 4.208. Complete the proof of Theorem 4.207 as follows:

1. Provide a map $g : q \to q'$ and show that $f_1 \fatsemi g = f_2 \fatsemi g$.
2. Show that g is a coequalizer of the pair f_1, f_2. ◊

4.4.3 Monoidal $*$-bifibration over Set

We will see that the functor $p \mapsto p(1)$ has special properties making it what **[shulman2008framed]** refers to as a *monoidal $*$-bifibration*. This means that **Set** acts as a sort of remote controller on the category **Poly**, grabbing every polynomial by its positions and pushing or pulling it this way and that.

For example, suppose one has a set A and a function $f : A \to p(1)$. Then we get a new polynomial f^*p with positions A. The notation here agrees with that in **??**: it is given by a pullback

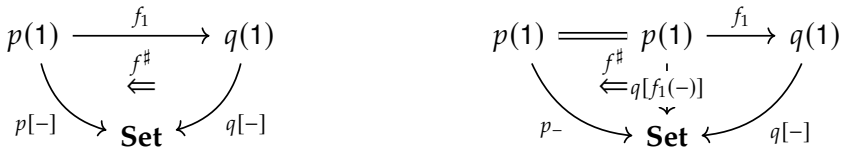
$$\begin{array}{ccc} f^*p & \xrightarrow{\text{cart}} & p \\ \downarrow & \lrcorner & \downarrow{\scriptstyle \eta_p} \\ A & \xrightarrow[f]{} & p(1) \end{array} \tag{4.209}$$

Here η_p is the unit of the adjunction $\mathbf{Set} \underset{p(1)}{\overset{A}{\rightleftarrows}} \mathbf{Poly}$. Since it is an isomorphismon positions and $p \mapsto p(1)$ is a right adjoint, the map $f^*p \to A$ is also an isomorphism on positions. Since on each position $a \in A$, the function $f_a^\sharp$ is an isomorphism on directions (both domain and codomain are the empty set), and since the a-position of f^*p is constructed by a pushout (see Example 4.204), each function $\mathrm{cart}_a^\sharp : p[f(a)] \to (f^*p)[a]$ is an isomorphism too. We'll see this as part of a bigger picture in Proposition 4.222 and Theorem 4.223.

Definition 4.210 (Vertical, cartesian)**.** Let $f : p \to q$ be a morphism of polynomials. It is called *vertical* if $f_1 : p(1) \to q(1)$ is an isomorphism. It is called *cartesian* if, for each $i \in p(1)$ the function $f_i^\sharp : q[f(i)] \to p[i]$ is an isomorphism.

Proposition 4.211. Every morphism in **Poly** can be uniquely factored as a vertical morphism followed by a cartesian morphism.

Proof. Recall from Eq. (4.59) that a morphism in **Poly** can be written as to the left; we can thus rewrite it as to the right:



The object $\left(\begin{smallmatrix} q[f_1 i] \\ i \in p(1) \end{smallmatrix}\right)$ is clearly unique up to isomorphism. □

Proposition 4.212. Vertical morphisms satisfy 2/3: given $p \xrightarrow{f} q \xrightarrow{g} r$ with $h = g \circ f$, if any two of f, g, h are vertical, then so is the third.

If g is Cartesian then h is cartesian iff f is cartesian.

Exercise 4.213. Give an example of polynomials p, q, r and maps $p \xrightarrow{f} q \xrightarrow{g} r$ such that f and $g \circ f$ are cartesian but g is not. ◊

Proposition 4.214. A morphism $f : p \to q$ is cartesian iff it is cartesian as a natural transformation, i.e. for any sets A, B and function $g : A \to B$, the naturality square

$$\begin{array}{ccc} p(A) & \xrightarrow{f_A} & q(A) \\ {\scriptstyle p(g)}\downarrow & \lrcorner & \downarrow{\scriptstyle q(g)} \\ p(B) & \xrightarrow[g_A]{} & q(B) \end{array}$$

is a pullback.

Exercise 4.215. Prove Proposition 4.214. ◊

Proposition 4.216. The monoidal structures $+, \times$, and $\otimes$ preserve cartesian morphisms.

Proof. Suppose that $f : p \to p'$ and $g : q \to q'$ are cartesian.

A position of $p + q$ is a position $i \in p(1)$ or a position $j \in q(1)$, and the map $(f + g)^\sharp$ at that position is either $f_i^\sharp$ or $g_j^\sharp$; either way it is an isomorphism, so $f + g$ is cartesian.

A position of $p \times q$ (resp. of $p \otimes q$) is a pair $(i, j) \in p(1) \times q(1)$. The morphism $(f \times g)^\sharp_{i,j}$ is $f_i^\sharp + g_j^\sharp$ (resp. $f_i^\sharp \times g_j^\sharp$) which is again an isomorphism if $f_i^\sharp$ and $g_j^\sharp$ are. Hence $f \times g$ (resp. $f \otimes g$) is cartesian, completing the proof. □

Proposition 4.217. Let $f : p \to q$ be a morphism and $g : q' \to q$ a cartesian morphism. Then the pullback g' of g along p

$$\begin{array}{ccc} p \times_q q' & \longrightarrow & q' \\ \downarrow & \lrcorner & \downarrow g \\ p & \xrightarrow[f]{} & q \end{array}$$

is cartesian.

Proof. This follows from Example 4.204 since the pushout of an isomorphism is an isomorphism. □

Theorem 4.218 (Cartesian morphisms in **Poly** are exponentiable)**.** If $f : p \to q$ is cartesian then the functor $f^* : \mathbf{Poly}/q \to \mathbf{Poly}/p$ given by pulling back $q' \to q$ along f is a left adjoint:

$$\mathbf{Poly}/p \underset{f_*}{\overset{f^*}{\underset{\longrightarrow}{\overset{\longleftarrow}{\Rightarrow}}}} \mathbf{Poly}/q$$

Proof. Fix $e : p' \to p$ and $g : q' \to q$.

$$\begin{array}{ccc} p' & & q' \\ {\scriptstyle e}\downarrow & & \downarrow{\scriptstyle g} \\ p & \xrightarrow[f]{} & q. \end{array}$$

We need to define a functor $f_* : \mathbf{Poly}/p \to \mathbf{Poly}/q$ and prove the analogous isomorphism establishing it as right adjoint to f^*. We first establish some notation. Given a set Q and sets $(P'_i)_{i\in I}$, each equipped with a map $Q \to P'_i$, let $Q/\sum_{i\in I} P'_i$ denote the coproduct in $Q/\mathbf{Set}$, or equivalently the wide pushout of sets P'_i with apex Q. Then we have the following formula for f_*p', which we write in larger font for clarity:

$$f_*p' \coloneqq \sum_{j\in q(1)} \sum_{i' \in \prod_{i\in f_1^{-1}(j)} e_1^{-1}(i)} y^{q[j]/\sum_{i\in f_1^{-1}(j)} p'[i'(i)]}. \tag{4.219}$$

Again, $q[j]/\sum_{i\in f_1^{-1}(j)} p'[i'(i)]$ is the coproduct of the $p'[i'(i)]$, taken in $q[j]/\mathbf{Set}$. Since $p[i] \cong q[f(i)]$ for any $i \in p(1)$ by the cartesian assumption on f, we have the following chain of natural isomorphisms

$$\begin{aligned} (\mathbf{Poly}/p)(f^*q', p') &\cong \prod_{i\in p(1)} \prod_{\{j'\in q'(1)\,|\,g_1(j')=f_1(i)\}} \sum_{\{i'\in p'(1)\,|\,e_1(i')=i\}} (p[i]/\mathbf{Set})(p'[i'], p[i] +_{q[f(i)]} q'[j']) \\ &\cong \prod_{i\in p(1)} \prod_{\{j'\in q'(1)\,|\,g_1(j')=f_1(i)\}} \sum_{\{i'\in p'(1)\,|\,e_1(i')=i\}} (q[f(i)]/\mathbf{Set})(p'[i'], q'[j']) \end{aligned}$$

$$\cong \prod_{j\in q(1)} \prod_{\{j'\in q'(1)\,|\,g_1(j')=j\}} \prod_{\{i\in p(1)\,|\,f_1(i)=j\}} \sum_{\{i'\in p'(1)\,|\,e_1(i')=i\}} (q[j]/\mathbf{Set})(p'[i'], q'[j'])$$

$$\cong \prod_{j\in q(1)} \prod_{\{j'\in q'(1)\,|\,g_1(j')=j\}} \sum_{i'\in\prod_{i\in f_1^{-1}(j)} e_1^{-1}(i)} \prod_{i\in f_1^{-1}(j)} (q[j]/\mathbf{Set})(p'[i'(i)], q'[j'])$$

$$\cong \prod_{j\in q(1)} \prod_{\{j'\in q'(1)\,|\,g_1(j')=j\}} \sum_{i'\in\prod_{i\in f_1^{-1}(j)} e_1^{-1}(i)} (q[j]/\mathbf{Set})\Big(\sum_{i\in f_1^{-1}(j)} p'[i'(i)], q'[j']\Big)$$

$$\cong (\mathbf{Poly}/q)(q', f_*p')$$ □

Example 4.220. Let $p \coloneqq 2y^2$, $q \coloneqq y^2 + y^0$, and $f : p \to q$ the unique cartesian morphism between them. Then for any $e : p' \to p$ over p, (4.219) provides the following description for the pushforward f_*p'. We use the isomorphisms $p(1) \cong 2$ and $q(1) \cong 2$ to talk about the positions of p and q.

Over the $j = 2$ position (which has $q[2] = 0$), we have $\prod_{i\in f_1^{-1}(2)e_1^{-1}(i)} \cong 1$ is an empty product, and $q[2]/\sum_{i\in f_1^{-1}(2)} p'[i'(i)]$ is an empty sum (in $q_2/\mathbf{Set}$), so we get $y^0 \cong 1$.

Over the $j = 1$ position (which has $q[1] = 2$), we have $\prod_{i'\in f_1^{-1}(1)e_1^{-1}(i)} \cong e_1^{-1}(1)\times e_1^{-1}(2)$. For each element $i' \in e_1^{-1}(1)\times e_1^{-1}(2)$, the sets $p'[i'(1)]$ and $p'[i'(2)]$ are each the codomain of a map from $q[1] = p[1] = p[2] = 2$ coming from $e^\sharp$, and $q[1]/\sum_{i\in\{1,2\}} p'[i'(i)]$ is the pushout $p'[i'(1)] \leftarrow 2 \rightarrow p'[i'(2)]$. Let's call that pushout set $X_{i'}$. Then in sum we have

$$f_*p' \cong \Big(\sum_{i'\in e_1^{-1}(1)\times e_2^{-1}(2)} y^{X_{i'}}\Big) + 1.$$

Remark 4.221. The category **Poly** is not locally cartesian closed; for example the map $y \to 1$ is not exponentiable. Indeed, in the notation of Theorem 4.218, let $p \coloneqq y$ and $q \coloneqq 1$, and let $p' = q' \coloneqq y^2$ with $e : y^2 \to y$ either projection. We'll show that the formula $\mathbf{Poly}/p(f^*q', p') \cong^? \mathbf{Poly}/q(q', f_*p')$ is impossible to satisfy.

We have $f^*(y^2) \cong y^3$ and hence

$$\mathbf{Poly}/p(f^*q', p') \cong \mathbf{Poly}/y(y^2, y^3) \cong 3.$$

There is no possibility for f_*p' because

$$\mathbf{Poly}/q(q', f_*p') \cong \mathbf{Poly}(y^2, f_*p') \cong 2^{\sum_{i\in f_*p'(1)}(f_*p')_i}$$

will always be a power of 2, and 3 is not a power of 2.

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 polynomial maps respecting the isomorphisms with A.

Proposition 4.222 (Base change)**.** For any function $f : A \to B$, pullback f^* along f induces a functor $B.\mathbf{Poly} \to A.\mathbf{Poly}$, which we also denote f^*.

Proof. This follows from (4.205) with $q := A$ and $r := B$, since pullback of an iso is an iso. □

Theorem 4.223. For any function $f\colon A \to B$, the pullback functor f^* has both a left and a right adjoint

$$A.\mathbf{Poly} \underset{f_*}{\overset{f_!}{\underset{\Leftarrow}{\overset{\Rightarrow}{\longleftarrow f^* \longrightarrow}}}} B.\mathbf{Poly} \tag{4.224}$$

Moreover $\otimes$ preserves the op-Cartesian arrows, making this a monoidal $*$-bifibration in the sense of [**shulman2008framed**].

Proof. Let p be a polynomial with $p(1) \cong A$. Then the formula for $f_!p$ and f_*p are given as follows:

$$f_!p \cong \sum_{b\in B} y^{\left(\prod_{a\mapsto b} p[a]\right)} \qquad \text{and} \qquad f_*p \cong \sum_{b\in B} y^{\left(\sum_{a\mapsto b} p[a]\right)} \tag{4.225}$$

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 Proposition 4.56, namely that **Poly** is equivalent to the Grothendieck construction applied to the functor $\mathbf{Set}^{\text{op}} \to \mathbf{Cat}$ sending each set A to the category $(\mathbf{Set}/A)^{\text{op}}$. The fact that functions $f\colon A \to B$ induces an adjoint triple between $\mathbf{Set}/A$ and $\mathbf{Set}/B$, and hence between $(\mathbf{Set}/A)^{\text{op}}$ and $(\mathbf{Set}/B)^{\text{op}}$ explains the variance in (4.225) and simultaneously establishes the adjoint triple (4.224).

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 4.216, the monoidal product $\otimes$ preserves cartesian morphisms; thus we will have established the desired monoidal $*$-bifibration in the sense of [**shulman2008framed**] as soon as we know that $\otimes$ preserves op-cartesian morphisms.

Given f and p as above, the op-cartesian morphism is the morphism $p \to f_!p$ obtained as the composite $p \to f^*f_!p \to f_!p$ where the first map is the unit of the $(f_!, f^*)$ adjunction and the second is the cartesian morphism for $f_!p$. On positions $p \to f_!p$ acts as f, and on directions it is given by projection.

If $f\colon p(1) \to B$ and $f'\colon p'(1) \to B'$ are functions then we have

$$\begin{aligned} f_!(p) \otimes f'_!(p') &\cong \sum_{b\in B}\sum_{b'\in B'} y^{\left(\prod_{a\mapsto b} p[a]\right)\times\left(\prod_{a'\mapsto b'} p'[a']\right)} \\ &\cong \sum_{(b,b')\in B\times B'} y^{\left(\prod_{(a,a')\mapsto(b,b')} p[a]\times p[b]\right)} \\ &\cong (f_! \otimes f'_!)(p \otimes p') \end{aligned}$$

and the op-cartesian morphisms are clearly preserved since projections in the second line match with projections in the first. □

4.5 Summary and further reading

...

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

See work by Gambino and Kock, Joyal, Paul Taylor, Michael Abbott and Neil Ghani (containers), ...

Chapter 5

A different category of categories

We have seen that the category of polynomial functors—sums of representables **Set** → **Set** and the natural transformations between them—has quite a bit of well-interoperating mathematical structure. Further, it is an expressive way to talk about dynamical systems that can change their interface and wiring pattern based on their internal states.

In this chapter we will discuss a monoidal structure on **Poly** that is quite easy from the mathematical point of view—it is simply composition—but which is again remarkable both in terms of its semantics and the phenomena that emerge mathematically.

In particular, we will see that the comonoids for the composition monoidal structure on **Poly** are precisely categories. However, the morphisms are different—they are often called *cofunctors*—and so we get a second category $\mathbf{Cat}^\sharp$ of categories and cofunctors. But the core groupoids of each—the groupoid of small categories and all functor isomorphisms between them, as well as the groupoid of small categories and all cofunctor isomorphisms between them—are isomorphic as groupoids. In other words, the following slogan is justified:

Polynomial comonads are precisely categories.

Cofunctors are not too familiar, but we will explain how to think of them in a variety of ways. We will see that whereas a functor $\mathcal{C} \to \mathcal{D}$ gives a kind of "picture" of $\mathcal{C}$ inside $\mathcal{D}$, a cofunctor $\mathcal{C} \nrightarrow \mathcal{D}$ gives a kind of $\mathcal{D}$-shaped "crystallization" of $\mathcal{C}$, one that is intuitively more geometric, more like creating neighborhoods. We will see in **??** that there is another kind of morphism between comonoids, namely the bimodules, that are perhaps more familiar: they are the so-called *parametric right adjoints*, or in perhaps more friendly terms, *data migration functors* between copresheaf categories.

The plan for this chapter is to first introduce what is perhaps the most interesting monoidal structure on **Poly**, namely the composition product; we do so in Section 5.1. We'll give a bunch of examples and ways to think about it in terms that relate to dynamical systems and our work so far. Then in **??** we'll discuss comonoids in **Poly** and explain why they are categories in down-to-earth, set-theoretic terms. We will also discuss the morphisms between them.

Finally in **??** we will discuss the cofree comonoid construction that takes any polynomial and returns a category. We will show how it relates to decision trees, as one may see in combinatorial game theory.

5.1 The composition product

In **??** we saw that the category **Poly** of polynomial functors—otherwise known as dependent lenses—is a very well-behaved category in which to think about dynamical systems of quite a general nature.

But we left one thing—what in some sense is the most interesting part of the story—out entirely. 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?[1]

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 **??**—in which the polynomial $p \circ q$ represents "starting at a position i in p, choosing a direction in $p[i]$, landing at a position j in q, choosing a direction in $q[j]$, and then landing... somewhere."

The composition product has many surprises up its sleeve, as we'll see. We've told many of them to you already in Section 4.1.5. We won't amass them all here; instead, we'll take you through the story step by step. But as a preview, this chapter will get us into decision trees, databases, and more dynamics, and it's all based on $\circ$.

As in Eq. (4.52), we'll continue to denote polynomials with the following notation

$$p \cong \sum_{i \in p(1)} y^{p[i]}, \tag{5.1}$$

and refer to $p(1)$ as the set of positions, and for each $i \in p(1)$ we'll refer to $p[i]$ as the set of direction at position i.

5.1.1 Defining the composition product

We begin with the definition of composition product.

Proposition 5.2. Suppose $p, q \in$ **Poly** are polynomial functors $p, q\colon$ **Set** $\to$ **Set**. Then their composite $p \circ q$ is again a polynomial functor and we have the following isomorphisms

$$p \circ q \cong \sum_{i \in p(1)} \prod_{d \in p[i]} \sum_{j \in q(1)} \prod_{e \in q[j]} y.$$

[1] If you're thinking "What could be more boring?", don't forget that our job is to connect this to the material in **??**. If that payoff doesn't intrigue you, then this chapter is not for you.

Proof. We can rewrite Eq. (5.1) for p and q as

$$p \cong \sum_{i \in p(1)} \prod_{d \in p[i]} y \qquad \text{and} \qquad q \cong \sum_{j \in q(1)} \prod_{e \in q[j]} y.$$

For any set X we have $(p \circ q)(X) = p(q(X)) = p(\sum_j \prod_e X) = \sum_i \prod_d \sum_j \prod_e X$, so (**??**) is indeed the formula for their composite. To see this is a polynomial, we use Proposition 4.192, which says we can rewrite the $\prod \sigma$ in (**??**) as a $\sigma \prod$. The result

$$p \circ q \cong \sum_{i \in p(1)} \sum_{j_i \colon p[i] \to q(1)} y^{\sum_{d \in p[i]} q[j_i(d)]}, \tag{5.3}$$

(written slightly bigger for clarity) is clearly a polynomial. □

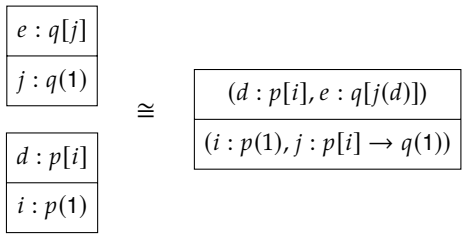
The composition of polynomials will be extremely important in the story that follows. However, we only sometimes think of it as composition; more often we think of it as a certain operation on arenas, or collections of corollas. Because we may wish to use $\circ$ to denote composition in arbitrary categories, we use a special symbol for polynomial composition namely

$$p \triangleleft q \coloneqq p \circ q.$$

The symbol $\triangleleft$ looks a bit like the composition symbol, in that it is an open shape, and when handwriting it fast, it's ok if it morphs into a $\circ$, but we'll soon see that it is quite evocative in terms of trees, and again it leaves $\circ$ for other uses.

We repeat the important formulas from **??** in the new notation:

$$p \triangleleft q \cong \sum_{i \in p(1)} \prod_{d \in p[i]} \sum_{j \in q(1)} \prod_{e \in q[j]} y. \tag{5.4}$$



Exercise 5.5. Let's consider (**??**) piece by piece, with concrete polynomials $p \coloneqq y^2 + y^1$ and $q \coloneqq y^3 + 1$.

1. What is q^2?
2. What is $y^2 \triangleleft q$?
3. What is $y^1 \triangleleft q$?
4. What is $(y^2 + y) \triangleleft q$? This is what $p \triangleleft q$ "should be".
5. How many functions $j_1 \colon p[1] \to q(1)$ are there?
6. For each function j_1 as above, what is $\sum_{d \in p[1]} q[j_1(d)]$?
7. How many functions $j_2 \colon p[2] \to q(2)$ are there?
8. For each function j_2 as above, what is $\sum_{d \in p[2]} q[j_2(d)]$?

9. Write out $\sum_{i \in p(1)} \sum_{j_i \colon p[i] \to q(1)} y^{\sum_{d \in p[i]} q[j_i(d)]}$.
10. Does the result agree with what $p \triangleleft q$ should be? ◊

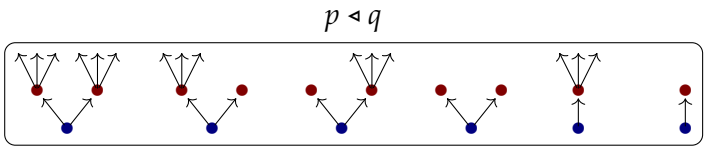
Exercise 5.6.

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. ◊

In terms of corollas, composition product $p \triangleleft q$ is given by glueing q-corollas onto the tips of p-corollas in every possible way. Let's say $p := y^2 + y$ and $q := y^3 + 1$, which as in Eq. (4.28) we draw as follows

(5.7)

Then their composite $p \triangleleft q$ would be drawn like so:



It 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 read off that $p \triangleleft q$ is isomorphic to $y^6 + 3y^3 + 2$.

Exercise 5.8. Use p, q as in **??** and $r := y^2 + 1$ in the following.

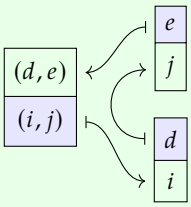
1. Draw $q \triangleleft p$.
2. Draw $p \triangleleft p$.
3. Draw $p \triangleleft p \triangleleft 1$.
4. Draw $r \triangleleft r$.
5. Draw $r \triangleleft r \triangleleft r$. ◊

Exercise 5.9. Let A and B be arbitrary sets, and let p be an arbitrary polynomial. Which of the following isomorphisms exist?

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. $Ay \otimes p \cong Ay \triangleleft p$?
5. $y^A \otimes p \cong y^A \triangleleft p$?
6. $p \otimes Ay \cong p \triangleleft Ay$?

7. $p \otimes y^A \cong p \triangleleft y^A$? ◊

Example 5.10. For any p and q there is an interesting map $o_{p,q} : p \otimes q \to p \triangleleft q$ that orders the operation. It looks like this:



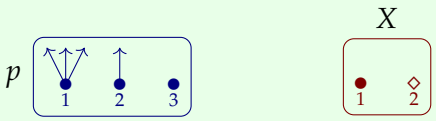
In other words, $p \triangleleft q$ is allowed to have j depend on d, whereas $p \otimes q$ is not; the map is in some sense the inclusion of the order-independent part. And of course we can flip the order using the symmetry $q \otimes p \cong p \otimes q$. This is, we just as well have a map $p \otimes q \to q \triangleleft p$.

Both $\otimes$ and $\triangleleft$ have the same monoidal unit, the identity functor y, and the identity is the unique map $y \to y$. The maps $o_{p,q}$ should commute with associators and unitors.

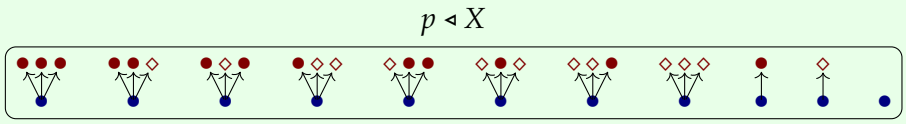
This can be used in the following way. Maps $p \to q \triangleleft r$ into composites are fairly easy to diagram and understand, whereas maps $q \triangleleft r \to p$ are not so easy to think about. However, given such a map, one may always compose it with $o_{q,r}$ to obtain a map $q \otimes r \to p$; this is quite a bit simpler to think about, more like a wiring diagram.

Example 5.11. For any set X and polynomial p, we can take $p(X) \in \mathbf{Set}$; indeed $p : \mathbf{Set} \to \mathbf{Set}$ is a functor! In particular, by this point you've seen us write $p(1)$ hundreds of times. But we've also seen that X is itself a polynomial, namely a constant one.

It's not hard to see that $p(X) \cong p \triangleleft X$. Here's a picture, where $p := y^3 + y + 1$ and $X := 2$.



Let's see how $(y^3 + y + 1) \triangleleft 2$ looks.



It has 11 positions and no open leaves, which means it's a set (constant polynomial), namely $p \triangleleft X \cong 11$.

We could also draw $X \triangleleft p$, since both are perfectly valid polynomials. Here it is:

$X \triangleleft p$

Each of the open leaves in X—of which there are none—is filled with a corolla from p.

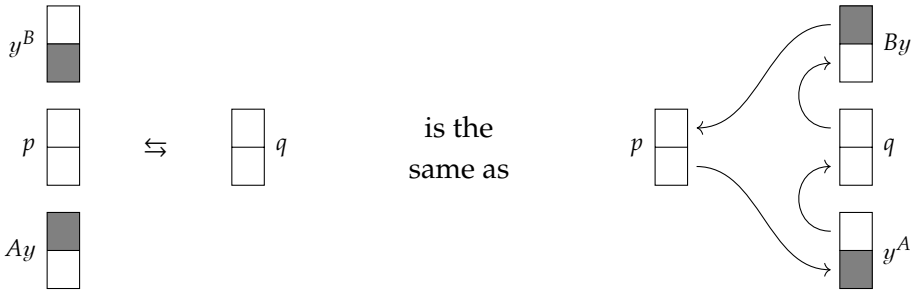
Exercise 5.12.

1. Choose a polynomial p and draw $p \triangleleft 1$ in the style of **??**.
2. Is it true that if X is a set (considered as a constant polynomial) and p is any polynomial, then $X \triangleleft p \cong X$?
3. Is it true 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? ◊

Proposition 5.13. For all sets A, B, we have the following adjunction:

$$\mathbf{Poly}\left(Ay \triangleleft p \triangleleft y^B, q\right) \cong \mathbf{Poly}\left(p, y^A \triangleleft q \triangleleft By\right)$$

Moreover, this isomorphism is natural in $A \in \mathbf{Set}^{\mathrm{op}}$ and $B \in \mathbf{Set}$.



Do you see how polyboxes with a black (one-element) part can flip upside-down to go to the other side?

Proof. Fill. □

Exercise 5.14. Let $A, B \in \mathbf{Set}$ be sets, and let $p \in \mathbf{Poly}$ be a polynomial. Is it true that the morphisms $Ay^B \to p$ can be identified with the morphisms $A \to p \triangleleft B$, i.e. that there is a bijection:

$$\mathbf{Poly}(Ay^B, p) \cong^? \mathbf{Poly}(A, p \triangleleft B) \tag{5.15}$$

If so, why? If not, give a counterexample. ◊

Exercise 5.16. For any $p \in \mathbf{Poly}$ there are natural isomorphisms $p \cong p \triangleleft y$ and $p \cong y \triangleleft p$.

1. Thinking of polynomials as functors $\mathbf{Set} \to \mathbf{Set}$, what functor does y represent?
2. Why is $p \cong y$ isomorphic to p?
3. In terms of tree pictures, draw $y \triangleleft p$ and $p \triangleleft y$, and explain pictorially how to see the isomorphisms $y \triangleleft p \cong p \cong p \triangleleft y$. ◊

5.1.2 Monoidal structure $(\mathbf{Poly}, \triangleleft, y)$

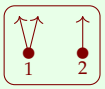
The technical claim is that $\triangleleft$ is a monoidal product, which means that it's well-behaved, in particular it's functorial, associative, and unital. In fact, all of this comes from general theory: for any category $\mathcal{C}$ the category whose objects are functors $\mathcal{C} \to \mathcal{C}$ and whose morphisms are natural transformations is a monoidal category. For us $\mathcal{C} := \mathbf{Set}$, and we are only using polynomial functors, not all functors, so there is a tiny bit to do, but it's accomplished by **??** and the fact that the identity functor $\mathbf{Set} \to \mathbf{Set}$ is a polynomial (it's y).

However, even though the formal theory of functors and natural transformations knocks the monoidality of $\triangleleft$ out of the park, it is still useful to discuss how it acts on morphisms in terms of positions and directions.

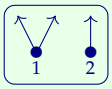
For any $f : p \to p'$ and $g : q \to q'$, we want to define a morphism $(f \triangleleft g) : (p \triangleleft q) \to (p' \triangleleft q')$. This is actually quite an impressive operation! It threads back and forth in a fascinating way.

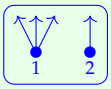
Recall from Example 4.66 that we can think of $f = (f_1, f^\sharp)$ as a way to delegate decisions from p to p'. Every decision (corolla / position) $i \in p(1)$ is assigned a decision $f_1(i) \in p'(1)$. Then every option $d \in p'[f_1(i)]$ there is passed back to an option $f^\sharp(d) \in p[i]$. Let's start with an example and then give the general method.

Example 5.17. Let's take $p := y^2 + y$, $q := y^2 + y$, $p' := y^3 + y$, and $q' := y + 1$.

$q =$ 

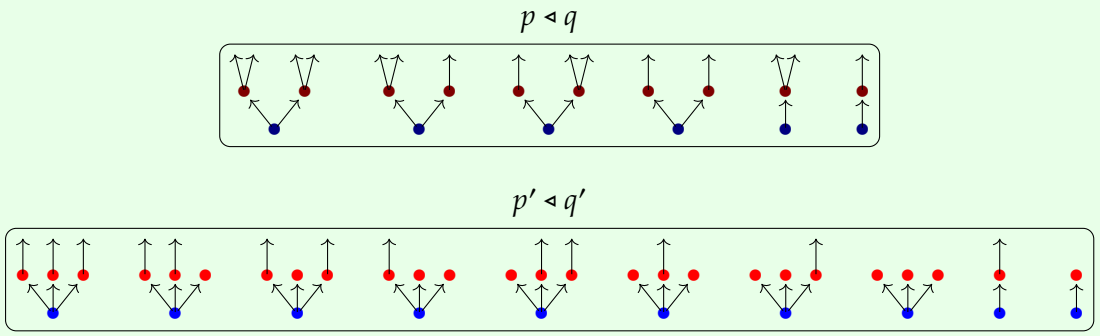
$q' =$ 

$p =$ 

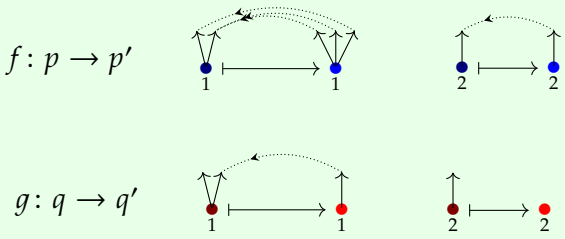
$p' =$ 

For any way to delegate from p to p' and q to q', we're supposed to give a way to

delegate from $(p \triangleleft q)$ to $(p' \triangleleft q')$. Let's draw $p \triangleleft q$ and $p' \triangleleft q'$.

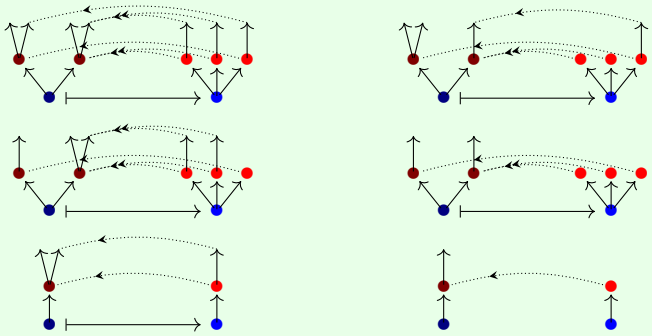


Ok, now suppose someone gives us delegations (morphisms / dependent lenses) $f : p \to p'$ and $g : q \to q'$. Let's just pick something relatively at random



Then we can form the induced delegation (morphism / dependent lens) $f \triangleleft g : (p \triangleleft q) \to (p' \triangleleft q')$ as follows. For each two-level tree (position in $p \triangleleft q$), we begin by using f to send the p-corolla on the bottom to a p'-corolla. The second-level nodes (from q') have not been chosen yet, but each of the p'-directions is passed back to a p-direction via $f^\sharp$. Now we use g to send the q-corolla at the second level to a q' corolla (this part is not shown in the diagram below because it would add clutter). Again each of the q'-directions is passed back to a q direction via $g^\sharp$.

Our six pictures below leave out the fact that the red corollas on the right are selected according to g; hopefully the reader can put it together for themselves.



Again, we're not making up these rules; it's a tree representation of how natural transformations f and g compose to form $f \triangleleft g$.

Exercise 5.18. With p, q, p', q' and f, g as in **??**, draw $g \triangleleft f : (q \triangleleft p) \to (q' \triangleleft p')$ in terms of trees as in the example. ◊

Exercise 5.19. Suppose p, q, and r are polynomials and you're given arbitrary morphisms $f : q \to p \triangleleft q$ and $g : q \to q \triangleleft r$. Does the following diagram necessarily commute?

$$\begin{array}{ccc} q & \xrightarrow{\quad g \quad} & q \triangleleft r \\ {\scriptstyle f}\downarrow & ? & \downarrow{\scriptstyle f \triangleleft r} \\ p \triangleleft q & \xrightarrow[p \triangleleft g]{} & p \triangleleft q \triangleleft r \end{array}$$

That is, do we have $(p \triangleleft g) \circ f =^{?} (f \triangleleft r) \circ g$? ◊

Pronouncing polynomial composites. We want to be able to pronounce polynomials like p and q in some way, which is intuitive and which lends itself to pronouncing composites like $p \triangleleft q$ or $p \triangleleft p \triangleleft q \triangleleft p$. We pronounce the polynomial

$$p = \sum_{i \in p(1)} \prod_{d \in p[i]} y$$

as "a choice of p-position i and, for every direction $d \in p[i]$ there, a future." Other than the word "future" in place of y, this is just pronouncing dependent sums and products. By saying "a future", we indicate that y is a functor: for any set X one could put in its place, we'll get an element of that X. We know we're getting an element of something, we just don't yet know what.

To pronounce composites of polynomials $p \triangleleft q$, we pronounce almost all of p, except we replace "future" with q. More precisely, to pronounce $p \triangleleft q$, which has the formula

$$p \triangleleft q \cong \sum_{i \in p(1)} \prod_{d \in p[i]} \sum_{j \in q(1)} \prod_{e \in q[j]} y,$$

we would say "a choice of position $i \in p(1)$ and, for every direction $d \in p[i]$ there, a choice of position $j \in q(1)$ and, for every direction $e \in q[j]$ there, a future.

Exercise 5.20.

1. Let p be an arbitrary polynomial. Write out the English pronunciation of $p \triangleleft p \triangleleft p$.
2. Pronouncing the unique element of 1 as "completion", write out the pronunciation of $p \triangleleft p \triangleleft 1$.
3. Pronouncing $\prod_{d \in \emptyset} y$ as "with no directions to travel, a complete dissociation from any purported future", write out the pronunciation of $p \triangleleft p \triangleleft y^0$.
4. With the "dissociation" language, pronounce $p \triangleleft 1 \triangleleft p$, and see if it makes sense with the fact that $p \triangleleft 1 \triangleleft p \cong p \triangleleft 1$. ◊

For any $n \in \mathbb{N}$, let $p^{\triangleleft n}$ denote the n-fold $\triangleleft$ power of p, e.g. $p^{\triangleleft 3} := p \triangleleft p \triangleleft p$. In particular, $p^{\triangleleft 1} := p$ and $p^{\triangleleft 0} := y$. We might think of $p^{\triangleleft n}$ in terms of length-n strategies, in the sense of game theory, except that the opponent is somehow abstract, having no positions of its own. That is, we pronounce $p^{\triangleleft n}$

Exercise 5.21. Let $p, q \in \mathbf{Poly}$ be polynomials and $n \in \mathbb{N}$; say $n \geq 1$. Pronounce $(p \triangleleft q)^{\triangleleft(n+2)}$, using the exact phrase "and so on, n times, ending with". ◊

5.1.3 Working with composites

We need a way of talking about maps to composites.[2] The set of morphisms $p \to q_1 \triangleleft q_2 \triangleleft \cdots \triangleleft q_k$ has the following form:

$$\mathbf{Poly}(p,\ q_1 \triangleleft q_2 \triangleleft \cdots \triangleleft q_k) \cong \prod_{i \in p(1)} \sum_{j_1 \in q_1(1)} \prod_{e_1 \in q_1[j_1]} \sum_{j_2 \in q_2(1)} \cdots \prod_{e_k \in q_k[j_{k-1}]} \sum_{d \in p[i]} 1$$

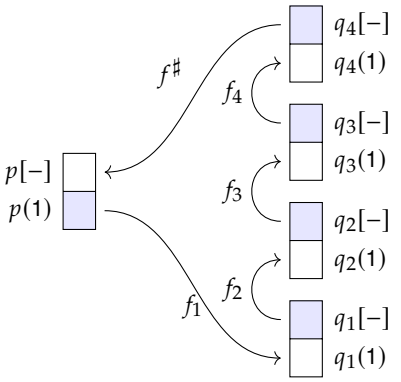
We can use this to generalize our notation in the case $k = 1$, i.e. for morphisms $p \to q$. That is we denoted such a morphism by $\binom{f^\sharp}{f_1}$, where $f_1 \colon p(1) \to q(1)$ and $f_i^\sharp \colon q[f_1(i)] \to p[i]$. We generalize this to the k-ary composite case as

$$(f_1, f_2, \ldots, f_k, f^\sharp) \colon p \longrightarrow q_1 \triangleleft q_2 \triangleleft \cdots \triangleleft q_k, \tag{5.22}$$

where

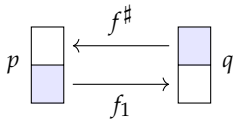
$$\begin{aligned}
&f_1 : p(1) \to q_1(1),\\
&f_2 : (i \in p(1)) \to (e_1 \in q_1[f_1(i)]) \to q_2(1),\\
&f_3 : (i \in p(1)) \to (e_1 \in q_1[f_1(i)]) \to (e_2 \in q_2[f_2(i, e_1)]) \to q_3(1),\\
&f_k : (i \in p(1)) \to (e_1 \in q_1[f_1(i)]) \to \cdots \to (e_{k-1} \in q_{k-1}[f_{k-1}(i, e_1, \ldots, e_{k-2})]) \to q_k(1),\\
&f^\sharp : (i \in p(1)) \to (e_1 \in q_1[f_1(i)]) \to \cdots \to (e_k \in q_k[f_k(i, e_1, \ldots, e_{k-1})]) \to p[i]
\end{aligned} \tag{5.23}$$

Here's a picture for the $k = 4$ case:

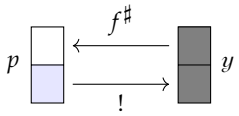


[2] It would be nice to also have a nice way of talking about maps *out of* composites; however that is more difficult.

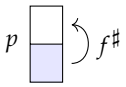
A few considerations might make (**??**) less scary. First of all, we're usually interested in the cases $k = 0, 1, 2$. The case $k = 1$ is just our $(f_1, f^\sharp)$ notation



The case $k = 0$ says that a map $p \to y$ is just a function $f^\sharp \in \prod_{i \in p(1)} p[i]$

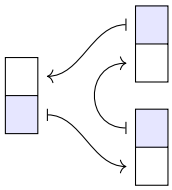


which we can rewrite simply as



(5.24)

The case $k = 2$ looks like this



(5.25)

and is considered in **??**. But before we get there, let's think about (**??**) in terms of delegating decisions by "due process".

Suppose we're p and someone gives us a decision $i \in p(1)$ to make: we're supposed to pick an element of $p[i]$. Luckily, by virtue of our morphism $f : p \to q_1 \triangleleft \cdots \triangleleft q_k$, consisting of steps $f_1, \ldots, f_k$ and an interpretation $f^\sharp$, we have a process to follow by which we can make the decision. Our first step is to ask q_1 to make decision $f_1(i)$. It dutifully chooses some option, say $e_1 \in q_1[f_1(i)]$. We know exactly what to do: our second step is to ask q_2 to make decision $f_2(i, e_1)$. It dutifully chooses some option, say $e_2 \in q_2[f_2(i, e_1)]$. We continue with the plan through step k, at which point we ask q_k to make decision $f_k(i, e_1, \ldots, e_{k-1})$. It dutifully chooses some option, say e_k, which we then interpret via our passback function $f^\sharp$ to obtain the desired decision $f^\sharp(i, e_1, \ldots, e_k) \in p[i]$.

A morphism $p \to q_1 \triangleleft \cdots \triangleleft q_k$ is a multi-step policy for p to make decisions by asking for decisions from q_1 then q_2, etc., all the way until q_k, and interpreting the results.

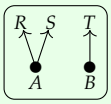
Example 5.26 (Maps $p \to q \triangleleft r$). By (**??**), a morphism $p \to q \triangleleft r$ can be specified by a tuple $(f_1, f_2, f^\sharp)$, where $f_1 : p(1) \to q(1)$ and $f_2, f^\sharp$ are a little more involved because they are dependent functions.

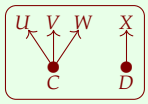
The dependent function f_2 takes as input a pair (i, d) where $i \in p(1)$ and $d \in q[f_1(i)]$,

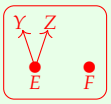
and it outputs an element of $r(1)$.

The dependent function $f^\sharp$ takes as input a tuple (i, d, e), where i, d are as above and $e \in r[f_2(i, d)]$, and it outputs an element of $p[i]$.

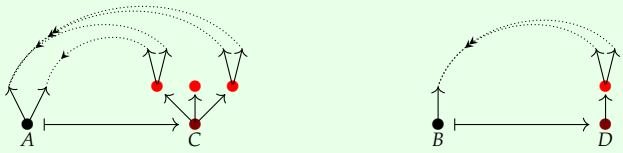
For example, let $p := \{A\}y^{\{R,S\}} + By^{\{T\}}$, $q := \{C\}y^{\{U,V,W\}} + \{D\}y^{\{X\}}$, and $r := \{E\}y^{\{Y,Z\}} + \{F\}$.

$p =$ 

$q =$ 

$r =$ 

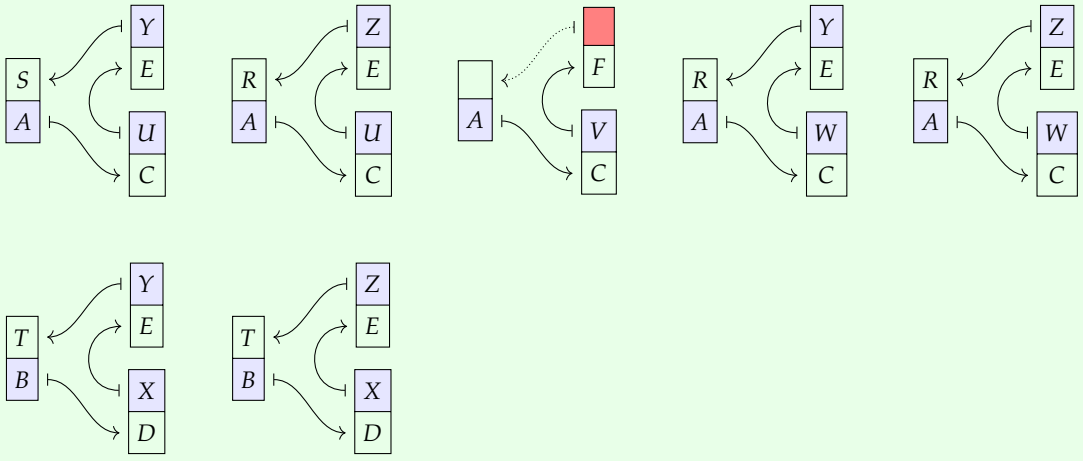
Here is a picture of a map $p \to q \triangleleft r$:



If we write it as $(f_1, f_2, f^\sharp)\colon p \to q \triangleleft r$ then we have

$$f_1(A) = C, \quad f_1(B) = D,$$
$$f_2(A, U) = E, \quad f_2(A, V) = F, \quad f_2(A, W) = E,$$
$$f_2(B, X) = E,$$
$$f^\sharp(A, U, Y) = S, \quad f^\sharp(A, U, Z) = R, \quad f^\sharp(A, W, Y) = R, \quad f^\sharp(A, W, Z) = R$$
$$f^\sharp(B, X, Y) = T, \quad f^\sharp(B, X, Z) = T$$

Box-notation uses a slightly different organization to represent the same data:



Example 5.27 (Using (**??**) to denote positions and directions in a composite). Suppose given polynomials $p_1, \ldots, p_k$. Recall from Exercise 4.173 that a position in their com-

posite as a map

$$i : y^1 \to p_1 \triangleleft \cdots \triangleleft p_k.$$

We can denote i in the notation (**??**) as $i = (i_1, \ldots, i_k)$, forgoing the input to i_1 because it is always $1 \in 1$ and also forgoing $f^\sharp$ because it is always the unique map to 1. Then in this notation

$$i_1 \in p_1(1), \quad i_2 : p_1[i_1] \to p_2(1), \quad i_3 : (d_1 \in p_1[i_1]) \to (d_2 \in p_2[i_2(d_1)]) \to p_3(1),$$
$$i_k : (d_1 \in p_1[i_1]) \to (d_2 \in p_2[i_2(d_1)]) \to \cdots (d_{k-1} \in p_{k-1}[i_{k-1}(d_1, \ldots, d_{k-2})]) \to p_k(1)$$

So for example to give a position in $p \triangleleft q \triangleleft r$ we need

$$i \in p(1), \quad j : p[i] \to q(1), \quad k : (d \in p[i]) \to (e \in q[j(d)]) \to r(1).$$

The direction-set of $p_1 \triangleleft \cdots \triangleleft p_k$ at position $(i_1, \ldots, i_k)$ is

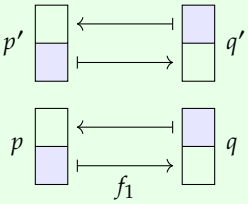
$$(p_1 \triangleleft \cdots \triangleleft p_k)[(i_1, \ldots, i_k)] \cong \sum_{d_1 \in p_1[i_1]} \sum_{d_2 \in p_2[i_2(d_1)]} \cdots \sum_{d_k \in p_k[i_k(d_1, \ldots, d_{k-1})]} 1$$

So for example given a position $(i, j, k) \in p \triangleleft q \triangleleft r$, a direction there consists of a tuple (d, e, f) where $d \in p[i]$, $e \in q[j(d)]$ and $f \in r[k(d, e)]$.

Exercise 5.28. Suppose $A_1, \ldots, A_k$ are sets and $p_i \coloneqq A_i y$ for each i. Use the notation of **??** to give the set of positions in $p \coloneqq p_1 \triangleleft \cdots \triangleleft p_k$. ◊

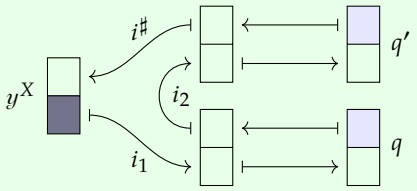
What if I give you a two-step decision to make of shape $p \triangleleft q$: I'll give you a position in p, you choose a direction there, and then based on your answer I'll give you a position in q, and you choose a direction there too. Now if each of p and q knew how to delegate its decisions to its partner, say $p \to p'$ and $q \to q'$, you should be able to delegate your two-step decision in $p \triangleleft q$ to a two-step decision by the partners $p' \triangleleft q'$. Here's how this looks in box-notation

Example 5.29 ($\triangleleft$ on morphisms). Given maps $f : p \to q$ and $f' : p' \to q'$, the corresponding map $(f \triangleleft f') : (p \triangleleft p') \to (q \triangleleft q')$ looks quite simple—even sterile—in box notation:



But it gets animated when someone chooses a map from an arbitrary representable (or

anything else); to do so is to choose a bunch of arrows as to the left:



One can now visualize the information flow through this sequence of delegations.

Exercise 5.30. Draw a picture analogous to (**??**) for the map $(f \triangleleft g \triangleleft h)\colon (p \triangleleft q \triangleleft r) \to (p' \triangleleft q' \triangleleft r')$, given $f = (f_1, f^\sharp)\colon p \to p'$, $g = (g_1, g^\sharp)\colon q \to q'$, and $h = (h_1, h^\sharp)\colon r \to r'$. Show what happens when one adds a map $y^X \to p \triangleleft q \triangleleft r$ from a representable. ◊

5.1.4 Mathematical aspects of $\triangleleft$

We now want to get at more subtle aspects of $p \triangleleft q$. We begin with the following.

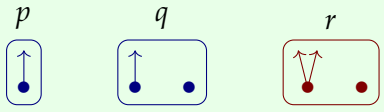
Proposition 5.31 (Left distributivity of $\triangleleft$)**.** For any polynomial r, the post-compose-with-r functor $(- \triangleleft r)\colon$ **Poly** $\to$ **Poly** commutes—up to natural isomorphism—with addition and multiplication:

$$(p + q) \triangleleft r \cong (p \triangleleft r) + (q \triangleleft r) \qquad \text{and} \qquad pq \triangleleft r \cong (p \triangleleft r)(q \triangleleft r).$$

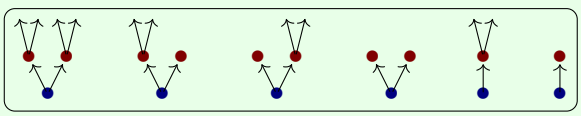
Proof. Formally, this just comes down to the fact that coproducts and products of functors **Set** $\to$ **Set** are computed pointwise and **Poly** is a full subcategory of $\mathbf{Set}^{\mathbf{Set}}$. One could instead give a proof in terms of $\sum$'s and $\prod$'s; this is done in **??**. □

Exercise 5.32. Prove **??** in terms of the formula for $\triangleleft$ given in **??**. ◊

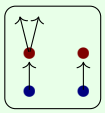
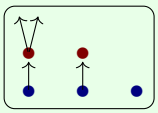
Example 5.33 (Picturing the left distributivity of $\triangleleft$ over $\times$). We want an intuitive understanding of this left-distributivity. Let $p := y$, $q := y + 1$, and $r := y^2 + 1$, as shown here:



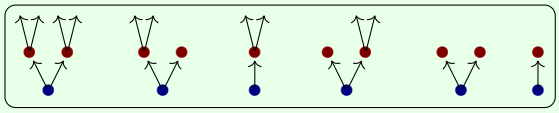
Then $pq \cong y^2 + y$ and we can draw $pq \triangleleft r$ as follows:

$(p \times q) \triangleleft r \cong$ 

Or we can compute $p \triangleleft r$ and $q \triangleleft r$ seperately:

$p \triangleleft r \cong$  $q \triangleleft r \cong$ 

and multiply them together by taking each tree from $p \triangleleft r$ and pairing it with each tree from $q \triangleleft r$:

$(p \triangleleft q) \times (p \triangleleft r) \cong$ 

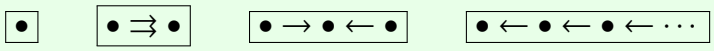
Exercise 5.34. Follow **??** with $+$ in place of $\times$: use pictures to give an intuitive understanding of the left-distributivity $(p + q) \triangleleft r \cong (p \triangleleft r) + (q \triangleleft r)$. ◊

Exercise 5.35. Show that the distributivities of **????** do not hold on the other side:

1. Find polynomials p, q, r such that $p \triangleleft (qr) \ncong (p \triangleleft q)(p \triangleleft r)$.
2. Find polynomials p, q, r such that $p \triangleleft (q + r) \ncong (p \triangleleft q) + (p \triangleleft r)$. ◊

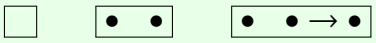
A connected limit is one whose indexing category J is (nonempty and) connected. That is, J has at least one object and any two objects are connected by a finite zigzag of arrows.

Example 5.36. 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.

Theorem 5.37 (Preservation of connected limits)**.** The operation $\triangleleft$ commutes with connected limits in both variables. That is, if J is a connected category, $p : J \to \mathbf{Poly}$ is a functor, and $q \in \mathbf{Poly}$ is a polynomial, then there are natural isomorphisms

$$\left(\lim_{j \in J} p_j\right) \triangleleft q \cong \lim_{j \in J} (p_j \triangleleft q) \qquad \text{and} \qquad q \triangleleft \left(\lim_{j \in J} p_j\right) \cong \lim_{j \in J} (q \triangleleft p_j)$$

Sketch of proof. The claim for the left variable follows as in the proof of **??**: limits of functors **Set** $\to$ **Set** are computed pointwise and **Poly** is a full subcategory of $\mathbf{Set}^{\mathbf{Set}}$ closed under limits. The claim for the right-hand variable comes down to the fact that polynomials are sums of representables; representable functors commute with all limits and sums commute with connected limits in **Set**. See [**kock2012polynomial**] for details. □

Exercise 5.38. Use **??** in the following.

1. Let p be a polynomial, thought of as a functor $p\colon$ **Set** $\to$ **Set**. Show that p preserves connected limits (of sets).
2. Show that for any polynomials p, q, r we have an isomorphism:
$$p \triangleleft (qr) \cong (p \triangleleft q) \times_{(p \triangleleft 1)} (p \triangleleft r)$$
3. Show that the distributivity $pq \triangleleft r \cong (p \triangleleft r)(q \triangleleft r)$ is a special case of **??**.
4. Show that for any set A and polynomials p, q, we have an isomorphism $A(p \triangleleft q) \cong (Ap) \triangleleft q$. ◊

While we're here, it will be helpful to record the following.

Proposition 5.39. For any polynomial $q \in$ **Poly**, tensoring with q (on either side) preserves connected limits. That is, if J is connected and $p\colon J \to$ **Poly** is a functor, then there is a natural isomorphism:
$$\left(\lim_{j \in J} p_j\right) \otimes q \cong \lim_{j \in J} (p_j \otimes q).$$

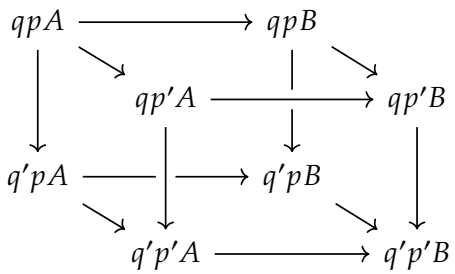
Proposition 5.40. For any polynomials p, p', q, q' there are natural maps
$$(p \triangleleft p') + (q \triangleleft q') \to (p + q) \triangleleft (p' + q') \tag{5.41}$$
$$(p \triangleleft p') \otimes (q \triangleleft q') \to (p \otimes pq) \triangleleft (p' \otimes q') \tag{5.42}$$
$$(p \triangleleft p') \times (q \triangleleft q') \leftarrow (p \times q) \triangleleft (p' \times q') \tag{5.43}$$
making $(+, \triangleleft)$ and $(\otimes, \triangleleft)$ duoidal structures and $(\times, \triangleleft)$ op-duoidal.

Proof. For (**??**) we have inclusion maps $p \to p + q$ and $p' \to p' + q'$, inducing a map $p \triangleleft p' \to (p + q) \triangleleft (p' + q')$. Similarly we obtain a map $q \triangleleft q' \to (p + q) \triangleleft (p' + q')$, so we get the desired map from the universal property of coproducts. It is straightforward to check that this is duoidal. The result for (**??**) is similar.

It remains to give a map (**??**).** □

Proposition 5.44 ($\triangleleft$ preserves cartesian maps in both variables)**.** If $f : p \to p'$ and $g : q \to q'$ are cartesian then so is $(f \triangleleft g) : (p \triangleleft q) \to (p' \triangleleft q')$.

Proof. For any $h : A \to B$ of sets, all faces of the cube



are pullbacks by Proposition 4.214 and **??**. Hence the diagonal is too by standard properties of pullbacks. □

Exercise 5.45.

1. Show that if f is an isomorphism and g is vertical then $f \triangleleft g$ is vertical.
2. Find a polynomial q and a vertical morphism $f : p \to p'$ such that $(f \triangleleft \mathrm{id}_q) : (p \triangleleft q) \to (p' \triangleleft q)$ is not vertical. ◊

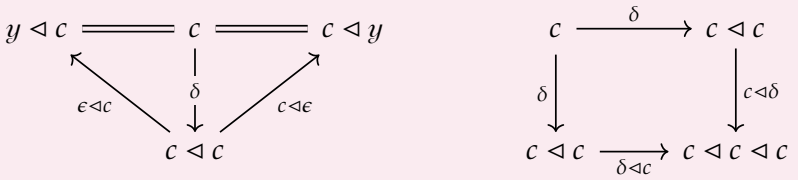
5.2 Comonoids in Poly

Imagine a sort of realm, where there are various positions you can be in. From every position, there are a number of moves you can make, possibly infinitely many. But whatever move you make, you'll end up in a new position. Well, technically it counts as a move to simply stay where you are, so you might end up in the same position. But wherever you move to, you can move again, and the any number of moves from an original place counts as a single move. What sort of realm is this?

The most surprising aspects of **Poly** really begin with its comonoids. In 2018, researchers Daniel Ahman and Tarmo Uustalu showed that comonoids in $(\mathbf{Poly}, y, \triangleleft)$ can be identified with categories. Every category in the usual sense is a comomonoid in **Poly** and every comonoid in **Poly** is a category. We find this revelation to be truly shocking, and suggests some very different ways to think about categories. Let's go through it.

Definition 5.46 (Comonoid)**.** A *comonoid* in a monoidal category $(\mathcal{C}, I, \triangleleft)$ is a tuple (c, ϵ, δ) where $c \in \mathcal{C}$ is an object, and $\epsilon : c \to I$ and $\delta : c \to c \triangleleft c$ are maps, such that the

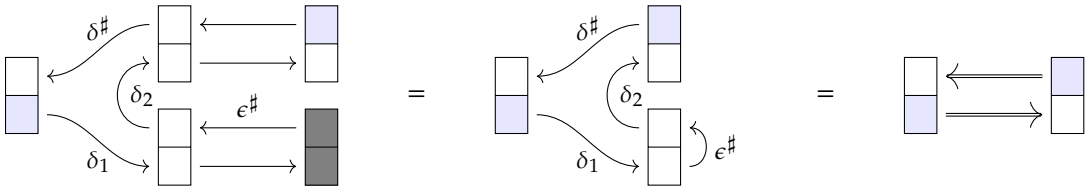
following diagrams commute:



(5.47)

We refer to a comonoid $P := (p, \epsilon, \delta)$ in $(\mathbf{Poly}, y, \triangleleft)$ as a *polynomial comonoid*.

Here's a picture of one of the unit laws:



We'll put the associativity and the other unitality picture in the following exercise. The meaning of ϵ and δ will become clear; for those who want a hint, see **??**.

Exercise 5.48.

1. Draw the other unitality equation.
2. Draw the associativity equation. ◊

Example 5.49 (δ^n notation). Let (c, ϵ, δ) be a comonoid. From the associativity of δ, the two ways to get a map $c \to c \triangleleft c \triangleleft c$ have the same result. This is true for any $n \in \mathbb{N}$: we get an induced map $c \to c^{\triangleleft n+1}$, which by mild abuse of notation we denote δ^n:

$$c \xrightarrow{\delta} c \triangleleft c \xrightarrow{c\triangleleft\delta} c \triangleleft c \triangleleft c \xrightarrow{c^{\triangleleft 2}\triangleleft\delta} \cdots \xrightarrow{c^{\triangleleft n}\triangleleft\delta} c^{\triangleleft(n+1)}.$$

In particular, we have $\delta^1 = \delta$ and we may write $\delta^0 := \mathrm{id}_c$ and $\delta^{-1} := \epsilon$.

Polynomial comonoids are usually called *polynomial comonads*. Though polynomials p can be interpreted as polynomial *functors* $p : \mathbf{Set} \to \mathbf{Set}$, we do not generally emphasize this part of the story; we use it when it comes in handy, but generally we think of polynomials more as dependent arenas, or sets of corollas.

Example 5.50 (The state comonad Sy^S). Let S be a set, and consider the polynomial $p := Sy^S$. It has a canonical comonoid structure—often called the *state* comonad—as we discussed in Section 4.3, page 199. To say it in the current language, we first need

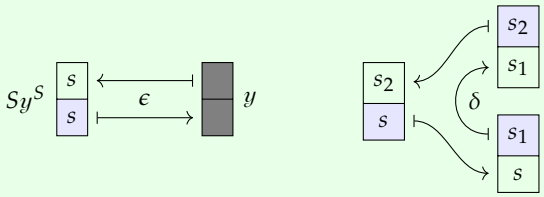
to give maps $\epsilon : p \to y$ and $\delta : p \to p \triangleleft p$. By **????**, this is equivalent to giving functions

$$S \xrightarrow{\epsilon'} S \qquad\qquad S \xrightarrow{\delta'} \sum_{s' \in S} \prod_{s_1 \in S} \sum_{s'_1 \in S} \prod_{s_2 \in S} S$$

We take ϵ' to be the identity and we take δ' to be

$$s \xmapsto{\epsilon'} s \qquad\qquad s \xmapsto{\delta'} (s' := s, s_1 \mapsto (s'_1 := s_1, s_2 \mapsto s_2)). \tag{5.51}$$

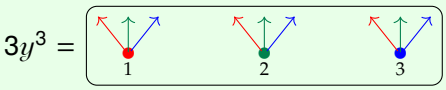
If you know how to read such things, you'll see that each element $s \in S$ is just being passed in a straightforward way. But we find this notation cumbersome and prefer the poly-box notation.



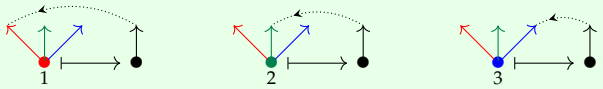
(5.52)

Exercise 5.53. Let $p := Sy^S$. For any $n \in \mathbb{N}$, write out the morphism of polynomials $\delta^n : p \to p^{\triangleleft(n+1)}$ either set-theoretically or in terms of poly-boxes as in **??** ◊

Example 5.54 (Picturing the comonoid Sy^S). Let's see this whole thing in pictures. First of all, let's take $S := 3 \cong \{\bullet, \bullet, \bullet\}$ and draw $\{\bullet, \bullet, \bullet\}y^{\{\bullet,\bullet,\bullet\}}$:

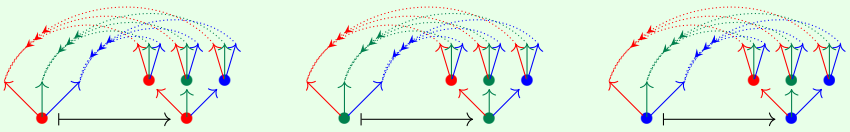


The map $\epsilon : Sy^S \to y$ can be drawn as follows:



It picks out one direction at each position, namely the one of the same color.

The map $\delta : Sy^S \to (Sy^S)^{\triangleleft 2}$ can be drawn as follows:



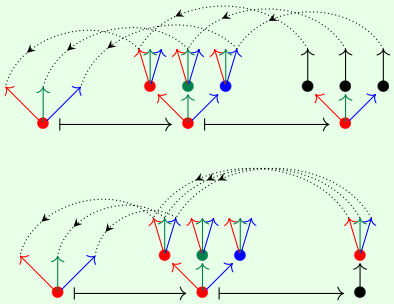
(5.55)

Note that $(Sy^S)^{\triangleleft 2}$ has SS^S, or in this case 81 many trees, only three of which are being pointed to by δ. That is, there is in general no rule on trees in that says the color of an arrow should agree in any sense with the color of the node it points to: (**??**) shows that the comonoid structure is pointing out the special trees where that does occur.

It remains to check the comonoid laws, the three commutative diagrams in (**??**). The first two say that the composites

$$Sy^S \xrightarrow{\delta} (Sy^S)^{\triangleleft 2} \xrightarrow{\mathrm{id}\triangleleft\epsilon} Sy^S \qquad \text{and} \qquad Sy^S \xrightarrow{\delta} (Sy^S)^{\triangleleft 2} \xrightarrow{\epsilon\triangleleft\mathrm{id}} Sy^S$$

are the identity. Let's return to the case $S = 3$. Then the second map in each case involves 81 different assignments, but only three of them will matter.[a] Since all three are strongly similar, we will draw only the red case. We also only draw the relevant passback maps.



We do not show associativity here, but instead leave it to the reader in **??**.

[a] To say technically that we can disregard all but three positions in $(Sy^S)^{\triangleleft 2} \cong SS^S y^{SS}$, one can use Proposition 4.211.

Exercise 5.56. Let $S := 2$ and $c := 2y^2$.

1. Draw c using color if possible.
2. We know c is supposed to be the carrier of a comonoid (c, ϵ, δ). Which two maps $c \to c^{\triangleleft 3}$ are supposed to be equal by associativity?
3. Draw these two maps in the style of **??**.
4. Are they equal? ◊

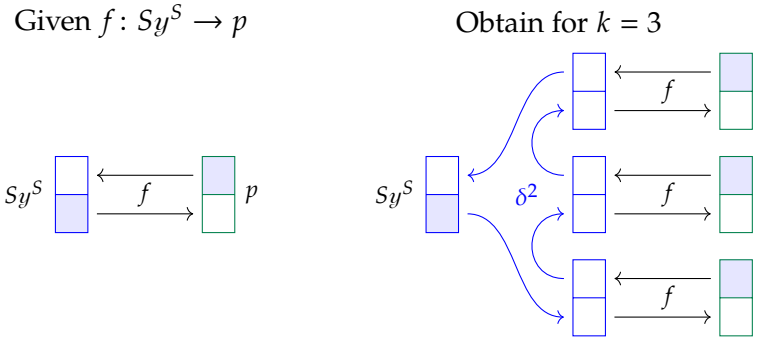
Speeding up dynamical systems Suppose we have a dynamical system $f : Sy^S \to p$, and we want to make it go k-times faster. That is, in every moment, we want it to process k-many inputs, rather than one.

Since Sy^S has the structure of a comonoid, we know that for every $k \in \mathbb{N}$ we have a map $\delta^{k-1} : Sy^S \to (Sy^S)^{\triangleleft k}$ by **??**. But we also have maps $f^{\triangleleft k} : (Sy^S)^{\triangleleft k} \to p^{\triangleleft k}$ because $\triangleleft$

is a monoidal product. Thus we can form the composite

$$Sy^S \xrightarrow{\delta^{k-1}} (Sy^S)^{\triangleleft k} \xrightarrow{f^{\triangleleft k}} p^{\triangleleft k} \qquad \text{Spdup}_k(f) \tag{5.57}$$

For every state $s \in S$, we now have a length-k strategy in p, i.e. a tree of height k in p, or explicitly a choice of p-position and, for every direction there, another p-position and so on k times. Here is a poly-box drawing for the $k = 3$ case:

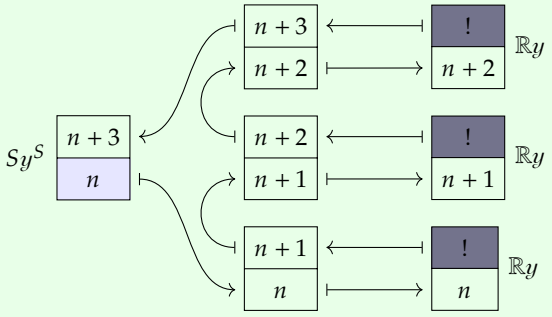


Example 5.58. Let $p := \mathbb{R}y^1$, let $S := \mathbb{N}$, and let $\begin{pmatrix} f^\sharp \\ f_1 \end{pmatrix} : Sy^S \to p$ be given by $f_1(n) := n$ and $f^\sharp(n, 1) := n + 1$. What is this speedup map $\text{Spdup}_k(f)$? First of all, its type is

$$\text{Spdup}_k(f) : Sy^S \to \mathbb{R}^k y,$$

meaning that it has the same set of states as before, but it outputs k-many reals in every moment.

So for example with $k = 3$ here is one moment of output:



So for example starting at initial state $n = 0$, we get the following output stream, e.g. for 4 seconds:

$$(0, 1, 2), (3, 4, 5), (6, 7, 8), (9, 10, 11).$$

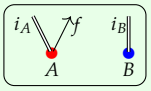
Other comonoids Once you know that these all important Sy^S-things are comonoids in **Poly**, it's interesting to ask "what are all the comonoids in **Poly**?" Let's discuss another one before answering the question in generality.

Example 5.59 (A simple comonoid that's not Sy^S). The polynomial $y^2 + y$ can be given a comonoid structure. Let's first associate names to its positions and directions.

Define $w \coloneqq \{A\}y^{\{i_A, f\}} + \{B\}y^{\{i_B\}}$; it is clearly isomorphic to $y^2 + y$, but its notation is meant to remind the reader of the walking arrow category

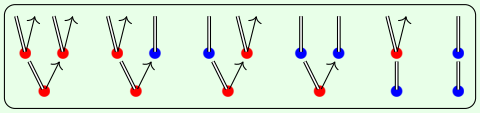
$$\mathcal{W} \coloneqq \boxed{A \xrightarrow{f} B}$$

We will use the category $\mathcal{W}$ as inspiration for equipping w with a comonoid structure (w, ϵ, δ). The map ϵ will pick out identity arrows and the map δ will tell us about codomains and composition (which is rather trivial in the case of $\mathcal{W}$). Here's a picture of $w \cong y^2 + y$:

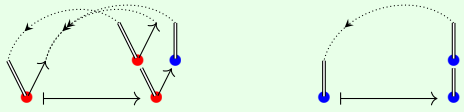
$w \coloneqq$ 

We first need to choose a map of polynomials $\epsilon\colon w \to y$; it can be identified with a dependent function $\epsilon^\sharp\colon (o \in w(1)) \to w[o]$, assigning to each position a direction there. Let's take $\epsilon^\sharp(A) \coloneqq i_A$ and $\epsilon^\sharp(B) \coloneqq i_B$:

Now we need a map of polynomials $\delta\colon w \to w \triangleleft w$. Let's draw out $w \triangleleft w$ to see what it looks like.

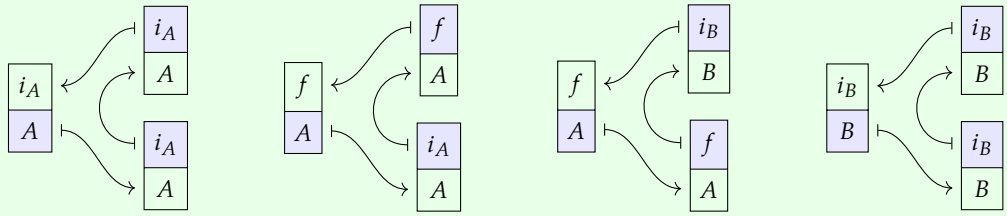
$w \triangleleft w =$ 

The map δ is going to tell us both about codomains and composition. Here it is:



The on-positions map selects, for each position (either A or B) the two-level tree starting at that position and having the correct codomains: the identity arrow on A points to the corolla for A; the f map points to the corolla for B; and the identity arrow on B points to the corolla for B. The on-directions maps assign the correct composites. Here

is $\delta : w \to w \triangleleft w$ again, in terms of poly-boxes.



It remains to check that (w, ϵ, δ) really is a comonoid, i.e. that the diagrams in (**??**) commute. We will check unitality only for A; it is easier for B.

In both pictures, one can see that the composite map is the identity. We would do associativity here, but because the category $\mathcal{W}$ is so simple, associativity is guaranteed; this makes the pictures too trivial.

Exercise 5.60. Write out the data (c, ϵ, δ) for the comonoid corresponding to the category

$$\boxed{B \xleftarrow{f} A \xrightarrow{g} C}$$

For this exercise, you are not being asked to check the unitality or associativity conditions. ◊

Exercise 5.61. Show that if A is a set and $p := Ay$ is the associated linear polynomial, then there exists a unique comonoid structure on p. ◊

5.2.1 Comonoids in Poly are categories

It turns out that comonoids in **Poly** are precisely categories. Strangely, however, the a morphism between comonoids is not a functor but something people are calling a *cofunctor*.

Definition 5.62 (Cofunctor)**.** Let $\mathcal{C}$ be a category with object set C_0, morphism set C_1, $\mathrm{dom}, \mathrm{cod} : C_1 \to C_0$ the domain and codomain,[a] and similarly for $\mathcal{D}$. A *cofunctor* $F : \mathcal{C} \nrightarrow \mathcal{D}$ consists of

1. a function $F : C_0 \to D_0$ *on objects* and
2. a function $F^\sharp : C_0 \times_{D_0} D_1 \to C_1$ *backwards on morphisms*,

satisfying the following conditions:

i. $F^\sharp(c, \mathrm{id}_{F_0 c}) = \mathrm{id}_c$ for any $c \in C_0$;
ii. $F_0 \operatorname{cod} F^\sharp(c, g) = \operatorname{cod} g$ for any $c \in C_0$ and $g \in D_{F_0(c)}$;
iii. $F^\sharp(\operatorname{cod} F^\sharp(c, g_1), g_2) \circ F^\sharp(c, g_1) = F^\sharp(c, g_1 \fatsemi g_2)$ for composable arrows g_1, g_2 out of $F_0 c$.

In other words, $F^\sharp$ preserves identities, codomains, and compositions.

We denote by $\mathbf{Cat}^\sharp$ the category of categories and cofunctors.

[a] We privilege the domain function $\mathrm{dom}\colon C_1 \to C_0$ in the sense that an unnamed map $C_1 \to C_0$ will be assumed to be dom.

The cofunctor laws can be written in commutative diagram form as follows:

$$\begin{array}{ccc} C_0 \times_{D_0} D_0 & \xrightarrow{\cong} & C_0 \\ {\scriptstyle \mathrm{id}_D}\downarrow & \text{(i)} & \downarrow{\scriptstyle \mathrm{id}_C} \\ C_0 \times_{D_0} D_1 & \xrightarrow[F^\sharp]{} & C_1 \end{array} \qquad \begin{array}{ccc} C_0 \times_{D_0} D_1 & \xrightarrow{F^\sharp} & C_1 \xrightarrow{\mathrm{cod}} C_0 \\ {\scriptstyle \pi_2}\downarrow & \text{(ii)} & \downarrow{\scriptstyle F_0} \\ D_1 & \xrightarrow[\mathrm{cod}]{} & D_0 \end{array}$$

$$\begin{array}{ccccc} C_0 \times_{D_0} D_1 \times_{D_0} D_1 & \xrightarrow{\fatsemi_D} & C_0 \times_{D_0} D_1 & \xrightarrow{F^\sharp} & C_1 \\ {\scriptstyle F^\sharp}\downarrow & & \text{(iii)} & & \uparrow{\scriptstyle \fatsemi_C} \\ C_1 \times_{D_0} D_1 & \xrightarrow[\cong]{} & C_1 \times_{C_0} C_0 \times_{D_0} D_1 & \xrightarrow[F^\sharp]{} & C_1 \times_{C_0} C_1 \end{array}$$

Theorem 5.63 (Ahman-Uustalu)**.** There is an equivalence of categories

$$\mathbf{Comon}(\mathbf{Poly}) \cong \mathbf{Cat}^\sharp.$$

Proof. This will be proved as **????**. □

Our first goal is to understand how one translates between categories $\mathcal{C}$ and comonoids $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ in **Poly**. The idea is pretty simple: the objects of $\mathcal{C}$ are the positions of $\mathfrak{c}$

$$\mathrm{Ob}(\mathcal{C}) \cong \mathfrak{c}(1)$$

and for each such object i, the morphisms $\{f : i \to j \mid j \in \mathrm{Ob}(\mathcal{C})\}$ emanating from i in $\mathcal{C}$ are the directions $\mathfrak{c}[i]$ there.

Definition 5.64. Let $\mathcal{C}$ be a category. The *emanation polynomial for* $\mathcal{C}$ is the polynomial

$$\mathfrak{c} := \sum_{i \in \mathrm{Ob}(\mathcal{C})} y^{\sum_{j \in \mathrm{Ob}(\mathcal{C})} \mathcal{C}(i,j)}$$

Exercise 5.65. What is the emanation polynomial for each of the following categories?

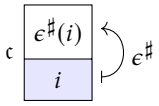
1. $\boxed{A \xrightarrow{f} B}$?
2. $\boxed{B \xleftarrow{f} A \xrightarrow{g} C}$?

3. The empty category?
4. The monoid $(\mathbb{N}, 0, +)$?
5. A monoid $(M, e, *)$?
6. The poset $(\mathbb{N}, \leq)$?
7. The poset $(\mathbb{N}, \geq)$? ◊

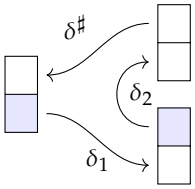
A category $\mathcal{C}$ is more than its emanation polynomial $\mathfrak{c}$, and a comonoid $(\mathfrak{c}, \epsilon, \delta)$ in **Poly** is more than its carrier polynomial $\mathfrak{c}$. The identities of $\mathcal{C}$ are all captured by the counit $\epsilon\colon \mathfrak{c} \to y$ and the codomain and composition information of $\mathcal{C}$ are all captured by the comultiplication map $\delta\colon \mathfrak{c} \to \mathfrak{c} \triangleleft \mathfrak{c}$. Our goal is to make this clear so that we can justly proclaim:

Comonoids in ***Poly*** *are precisely categories!*

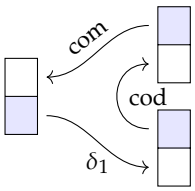
We want to understand how the counit ϵ and comultiplication δ in a comonoid $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ relate to identities, codomains, and composites in a category. We first use our work in **??** to get a better handle on ϵ and δ. For example, since $\epsilon\colon \mathfrak{c} \to y$ maps to the empty composite, we know by (**??**) that it is of the form



i.e. for every $i \in \mathfrak{c}(1)$, a choice of element $\epsilon^\sharp(i) \in \mathfrak{c}[i]$. Rather than call it $\epsilon^\sharp$, we will refer to this map as idy. Similarly, we know by (**??**) that δ is of the form

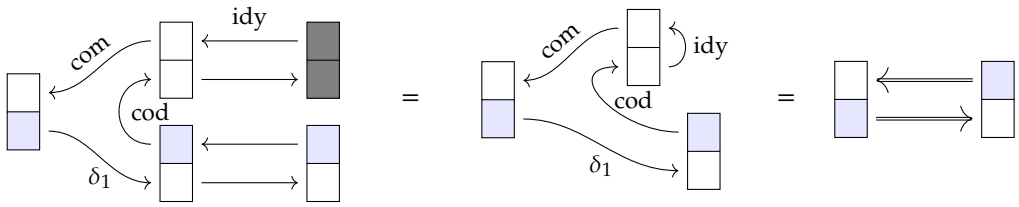


We've said that this secretly holds information about the codomains and composites for a category structure on $\mathfrak{c}$. How does that work? We will soon find that δ_1 is forced to be an identity, that δ_2 holds codomain information, and that $\delta^\sharp$ holds composite information. So we will use that notation here

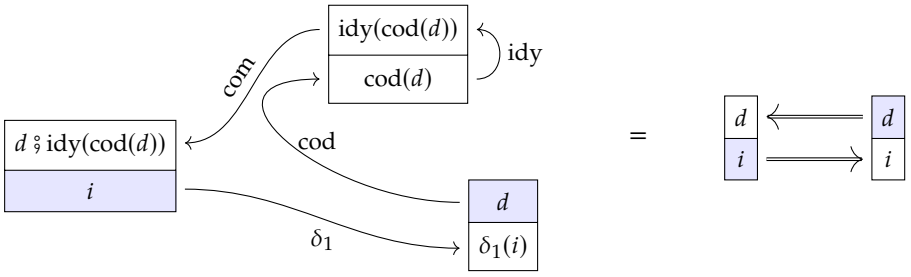


and our goal now is to see that the cod map really has something to do with codomains and that the com map really has something to do with composites, as advertised. What makes these true are the unitality and associativity equations required for $(\mathfrak{c}, \epsilon, \delta)$ to be a comonoid; see **??**.

To get started, we consider the first unitality equation from (**??**):

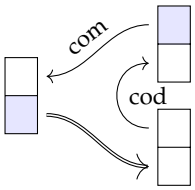


Let's add some arbitrary fillers $i \in \mathfrak{c}(1)$ and $d \in \mathfrak{c}[i]$ to the open slots, and hence obtain an equation:



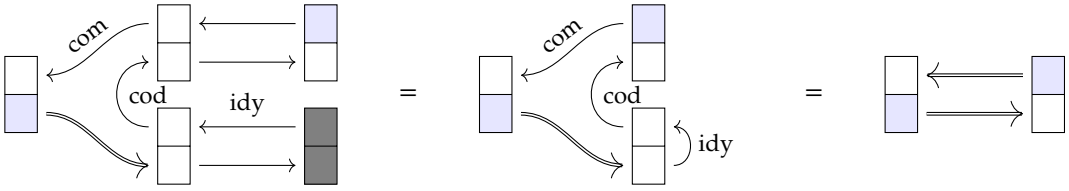
First it's saying that $\delta_1(i) = i$. This is great news; it means we can forget about δ_1, just as we said earlier. Second it's saying that $d \fatsemi \text{idy}(\text{cod}(d)) = d$. Unpacking, this means that composing a morphism d with the identity morphism on its codomain returns d. It's neat to watch the comonoid laws declaring the standard laws of categories. It's like meeting a like-minded toad; we never knew toads could be like-minded, but the phenomena don't lie.

Before moving on, we redraw $\delta : c \to c \triangleleft c$ with the information-lacking δ_1 (which the first unitality equation said was always identity) and replace it with a double arrow:

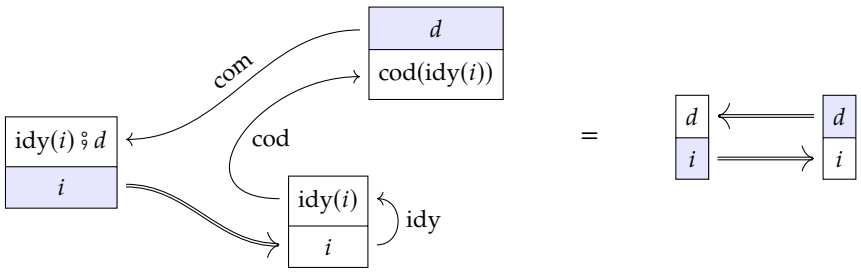


(5.66)

Now we can write the other unitality equation.

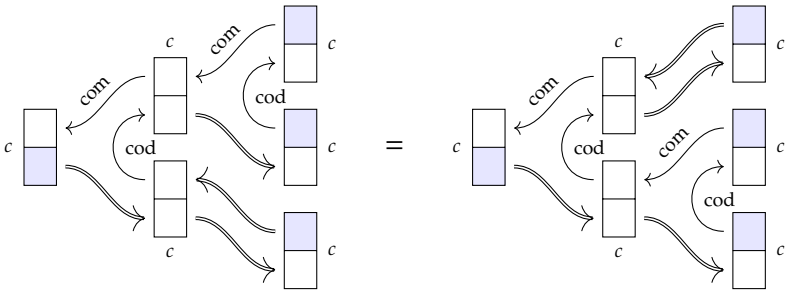


Let's add some arbitrary fillers $i \in c(1)$ and $d \in c[i]$ to get some equations:

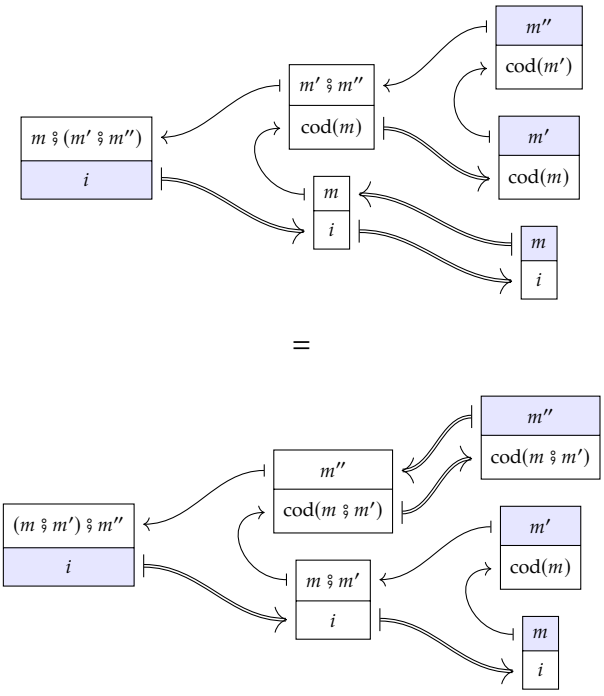


Ah, it's saying that it wants $\text{cod}(\text{idy}(i)) = i$, which makes sense—the codomain of the identity on i should be i—and that it wants $\text{idy}(i) \fatsemi d = d$, i.e. that composing the identity on i with d should return d. We couldn't have said it better ourselves; thanks like-minded toad!

Finally we draw the associativity equation.



Let's fill it in with $i \in c(1)$ and a sequence $i \xrightarrow{m} \xrightarrow{m'} \xrightarrow{m''}$ of emanating morphisms:



Ah, it's saying that it wants $\mathrm{cod}(m') = \mathrm{cod}(m \fatsemi m')$; well yeah, that's how codomains should work. And it wants $m \fatsemi (m' \fatsemi m'') = (m \fatsemi m') \fatsemi m''$, classic associativity. Amazing; thanks again toad!

We've seen that all of the data and equations of categories are embedded, though in a very non-standard way, in the data and equations of polynomial comonoids.

Exercise 5.67. Let $\mathcal{C}$ be a category, $\mathfrak{c}$ its emanation polynomial, and $i \in \mathrm{Ob}(\mathcal{C})$ an object. This exercise is for people who know the definition of the coslice category $i/\mathcal{C}$ of objects under i. Is it true that there is an isomorphism

$$\mathrm{Ob}(c/\mathcal{C})i \cong^{?} \mathfrak{c}[i]$$

If so, describe it; if not, give a counterexample. ◊

5.2.2 Examples showing the correspondence between comonoids and categories

Example 5.68 (Monoids). Let $(M, e, *)$ be a monoid. Then we can construct a comonoid structure on the representable y^M. A morphism $y^M \to y$ can be identified with an element of M; under that identification we take $\epsilon := e$. Similarly, $y^M \triangleleft y^M \cong y^{M^2}$ and a morphism $y^M \to y^{M^2}$ can be identified with a function $M^2 \to M$; under that identification we take $\delta := *$.

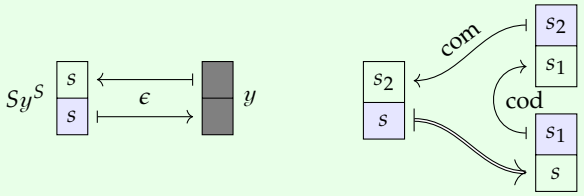
Exercise 5.69. Finish **??** by showing that if $(M, e, *)$ satisfies the unitality and associativity requirements of a monoid in $(\mathbf{Set}, 1, \times)$ then (y^M, ϵ, δ) satisfies the unitality and associativity requirements of a comonoid in $(\mathbf{Poly}, y, \triangleleft)$. ◊

Example 5.70 (What category is Sy^S?). The first comonoid we introduced, back in **??** was $\mathscr{S} = (p, \epsilon, \delta)$, where $p = Sy^S$ for some set S. Now we know that comonoids correspond to categories. So what category $\mathcal{S}$ corresponds to $\mathscr{S}$?

By the work above, we know that $\mathcal{S}$ has object set $S = p(1)$, and that for every object $s \in S$ there are S-many emanating morphisms, though we don't yet know their codomains nor the composition formula.

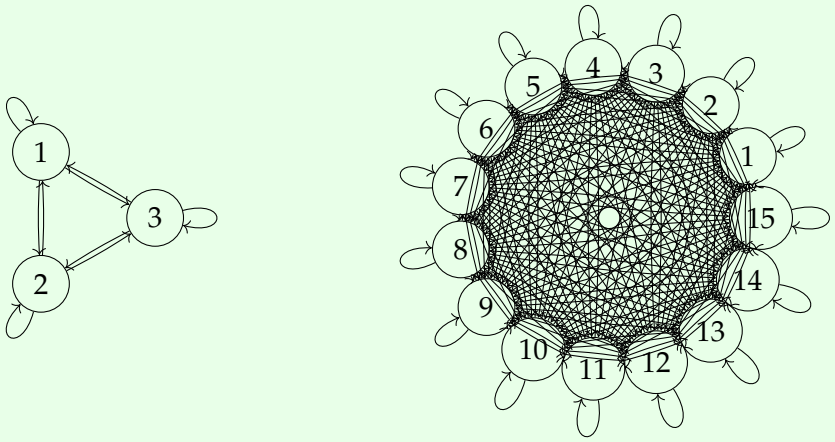
To calculate the codomains and compositions we examine the map $\delta \colon p \to p \triangleleft p$, which was given set-theoretically in (**??**) and in terms of poly-boxes in (**??**). We repeat

it here for your convenience:



The ϵ map is saying that the identity on the object s is the emanating morphism s. Remember that both the set of objects and the set of morphisms emanating from any given object are S. The map cod $= \delta_2$ is telling us that the codomain of the morphism s_1 emanating from s is the object s_1, and that the composite of s_1 and s_2 is $s_1 \fatsemi s_2 = s_2$.

What this all means is that $\mathcal{S}$ is the category with S-many objects and a unique morphism $s \to s'$ for any $s, s' \in S$. Here are pictures for $S = 3$ and $S = 15$, with all maps (even identities) drawn:



Some people would call this the contractible groupoid, or the terminal category with S-elements, or the unique category whose underlying graph is complete on S vertices. The one that's *least* good for us will be "terminal" category, because as we'll see, we're going to be interested in different sorts of morphisms between categories than the usual ones, namely cofunctors rather than functors, and $\mathcal{S}$ is not terminal for cofunctors.

Anyway, to avoid confusion, we'll refer to $\mathcal{S}$ as the *state category on* S, because we will use these to think about states of dynamical systems, and also because the state comonad in functional programming is Sy^S.

Exercise 5.71. We showed in **??** that for any set A, the linear polynomial $p := Ay$ has a unique comonoid structure. What category does it correspond to? ◊

Definition 5.72. Let $\mathcal{C}$ be a category and $c \in \mathrm{Ob}(\mathcal{C})$ an object. The *degree of* c, denoted $\deg(c)$ is the set of arrows in $\mathcal{C}$ that emanate from c.

If $\deg(c) \cong 1$, we say that c is *linear* and if $\deg(c) \cong \mathsf{n}$ for $n \in \mathbb{N}$, we say c has *degree* n.

Exercise 5.73.

1. If every object in $\mathcal{C}$ is linear, what does it mean about $\mathcal{C}$?
2. Is it possible for an object in $\mathcal{C}$ to have degree 0?
3. Find a category that has an object of degree $\mathbb{N}$.
4. How many categories are there that have just one linear and one quadradic (degree 2) object?
5. Is the above the same as asking how many comonad structures on $y^2 + y$ there are?

◊

Proposition 5.74. Let $F\colon \mathcal{C} \nrightarrow \mathcal{D}$ be a cofunctor, $c, c' \in \mathrm{Ob}(\mathcal{C})$ objects, and $g\colon F(c) \to F(c')$ a morphism in $\mathcal{D}$. Then if g is an isomorphism, so is $F_c^\sharp(g)$.

Proof. With $d \coloneqq F(c)$, $d' \coloneqq F(c')$, and g' the inverse of g, we have

$$\begin{aligned} \mathrm{id}_c &= F_c^\sharp(\mathrm{id}_d) \\ &= F_c^\sharp(g \fatsemi g') \\ &= F_c^\sharp(g) \fatsemi F_{c'}^\sharp(g') \end{aligned}$$

Thus $F_c^\sharp(g)$ is a section of $F_{c'}^\sharp(g')$. The opposite is true similarly, completing the proof. □

Exercise 5.75.

1. Find a category structure for the polynomial $y^{\mathsf{n}+1} + \mathsf{n}y$.
2. Would you call your category "star-shaped"? ◊

Exercise 5.76. Let S be a set. Is there any comonoid structure on Sy^S other than that of the state category? ◊

5.2.3 Morphisms of comonoids are cofunctors

Our next goal is to understand morphisms $\mathscr{C} \to \mathscr{D}$ between comonoids and what they look like as maps between categories $\mathcal{C} \to \mathcal{D}$.

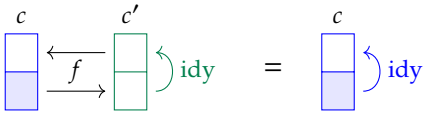
Cofunctors F are forward on objects, and backwards on morphisms. It's good to remember: Codomains are objects, so F preserves them going forwards; identities and composites are morphisms, so F preserves them going backwards.

Let's begin with a definition.

Definition 5.77 (Morphisms of comonoids)**.** Let $\mathscr{C} := (c, \epsilon, \delta)$ and $\mathscr{C}' := (c', \epsilon', \delta')$ be polynomial comonoids as in **??**. A *morphism* $\mathscr{C} \to \mathscr{C}'$ consists of a morphism $f : c \to c'$ of polynomials that commutes with the structure maps:

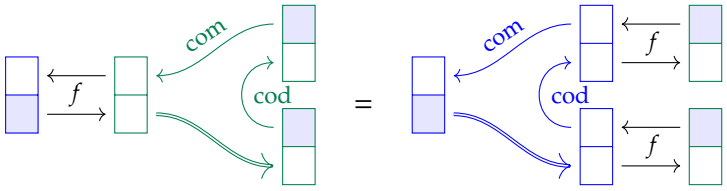
$$\begin{array}{ccc} c & \xrightarrow{f} & c' \\ {\scriptstyle\epsilon}\downarrow & & \downarrow{\scriptstyle\epsilon'} \\ y & = & y \end{array} \qquad\qquad \begin{array}{ccc} c & \xrightarrow{f} & c' \\ {\scriptstyle\delta}\downarrow & & \downarrow{\scriptstyle\delta'} \\ c \triangleleft c & \xrightarrow[f \triangleleft f]{} & c' \triangleleft c' \end{array} \tag{5.78}$$

Let's see the two laws of comonoid morphisms using poly-boxes. First the counit law:



(5.79)

Then the comultiplication law:



(5.80)

If we fill in **??** with an object $i \in \mathfrak{c}(1)$, we obtain the equation

$$f^\sharp(i, \mathrm{idy}(f_1(i))) = \mathrm{idy}(i),$$

which is the first law of **??**. If we fill in **??** with $i \in \mathfrak{c}(1)$ and $m \in \mathfrak{c}'[f(i)]$ and $m' \in \mathfrak{c}'[\mathrm{cod}(m)]$, we obtain the equations

$$\mathrm{cod}(m) = f_1\big(\mathrm{cod}\,\big(f^\sharp(i, m)\big)\big)$$
$$f^\sharp(i, \mathrm{com}(m, m')) = \mathrm{com}(f^\sharp(i, m), f^\sharp(\mathrm{cod}(f^\sharp(i, m)), m'))$$

and these are the second and third laws of **??**.

Exercise 5.81. Summarize the proof of **??**, developed above. You may cite anything written in the text so far. ◊

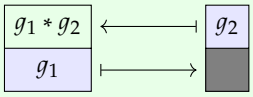
Examples of cofunctors. We saw in **??**, summarized in **??**, that cofunctors $\mathcal{C} \nrightarrow \mathcal{D}$ are the same thing as morphisms of comonoids $\mathscr{C} \to \mathscr{D}$, so we elide the difference. The question we're interested in now is: how do we think about cofunctors? What is a map of polynomial comonoids like?

The rough idea is that a cofunctor $\mathcal{C} \nrightarrow \mathcal{D}$ is, in particular, a morphism $\mathfrak{c} \to \mathfrak{d}$ in **Poly** between their emanation polynomials. This map preserves identities, codomains, and composition, which is great, but you still feel like you've got a map of polynomials on your hands: it goes forwards on objects and backwards on morphisms.

If a functor $\mathcal{C} \to \mathcal{D}$ is a picture *of $\mathcal{C}$ in $\mathcal{D}$, then a cofunctor $\mathcal{C} \nrightarrow \mathcal{D}$ is a D-*shaped crystallization *of C.*

Let's look at some examples to see how cofunctors look like crystallizations, or perhaps partitions.

Example 5.82. Let $(G, e, *)$ be a group and (y^G, ϵ, δ) the corresponding comonoid. There is a cofunctor $Gy^G \to y^G$ given by



To see this is a cofunctor, we check that identities, codomains, and compositions are preserved. For any g_1, the identity e is passed back to $g_1 * e = g_1$, and this is the identity on g_1 in Gy^G. Codomains are preserved because there is only one object in y^G. Composites are preserved because for any g_2, g_3, we have $g_1 * (g_2 * g_3) = (g_1 * g_2) * g_3$.

Exercise 5.83. Does the idea of **??** work when G is merely a monoid, or does something go subtly wrong somehow? ◊

Proposition 5.84. There is a fully faithful functor $\mathbf{Mon}^{\mathrm{op}} \to \mathbf{Cat}^\sharp$, whose image is precisely those categories whose emanation polynomial is representable.

Proof. Given a monoid $(M, e, *)$, we think of it as a category with one object; its emanation polynomial y^M is representable. A cofunctor between such categories carries no data in its on-objects part, and codomains are automatically preserved. Cofunctors $y^M \to y^N$ simply carry elements of N to elements of M, preserving identity and composition, exactly the description of monoid homomorphisms. □

Proposition 5.85. There is an adjunction

$$\mathbf{Cat}^\sharp(\mathcal{C}, Ay) \cong \mathbf{Set}(\mathrm{Ob}(\mathcal{C}), A)$$

where $\mathcal{C} \in \mathbf{Cat}^\sharp$ is a comonoid and $A \in \mathbf{Set}$ is a set.

Example 5.86. Consider the category $\mathbb{R}y^{\mathbb{R}}$, where the codomain of r emanating from x is $x + r$, identities are 0, and composition is given by addition. What are cofunctors into $\mathbb{R}y^{\mathbb{R}}$?

Let $\mathcal{C}$ be a category and $|\cdot| \colon \mathcal{C} \nrightarrow \mathbb{R}y^{\mathbb{R}}$ a cofunctor. It assigns to every object c both a real number $|c| \in \mathbb{R}$ and a choice of emanating morphism $|c|^\sharp(r) \colon c \to c_r$ such that $|c| + r = |c_r|$. This assignment satisfies some laws. Namely we have $c_0 = c$ and, given reals $r, s \in \mathbb{R}$, we have $(c_r)_s = c_{r+s}$.

Exercise 5.87.

1. Do you think a cofunctor $\mathcal{C} \nrightarrow \mathbb{R}y^{\mathbb{R}}$ as in **??** should be called an $(\mathbb{R}, 0, +)$-action on the objects of $\mathcal{C}$, or a filtration, or a valuation, or something else?
2. Why? ◊

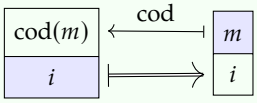
Exercise 5.88.

1. Over two discrete objects $\{A, B\}$, how many cofunctors are there

$$y^2 + y \cong \boxed{A \to B} \longrightarrow \boxed{A \rightrightarrows B} \cong y^3 + y$$

   from the walking arrow category to the walking parallel-arrows category?
2. What is meant more precisely by "over two discrete objects $\{A, B\}$" above? ◊

Exercise 5.89. Let $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ be a comonoid in **Poly**. We have a state category $\mathfrak{c}(1)y^{\mathfrak{c}(1)}$ on the set of objects of $\mathscr{C}$. There is a map of polynomials $\mathfrak{c}(1)y^{\mathfrak{c}(1)} \to \mathfrak{c}$ given by



for an object $i \in \mathfrak{c}(1)$ and an outgoing morphism $m \in \mathfrak{c}[i]$. Is this map a cofunctor? ◊

Example 5.90 (Canonical cofunctors from state categories). Let $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ be a comonoid, where $\delta = (\mathrm{id}, \mathrm{cod}, \mathrm{com})$ as in (**??**). For any position $i \in \mathfrak{c}(1)$, there is a cofunctor

$$(\mathrm{cod}, \mathrm{com}) \colon \mathfrak{c}[i]y^{\mathfrak{c}[i]} \nrightarrow \mathfrak{c}.$$

That is, an object $f \in \mathfrak{c}[i]$ is also a morphism in $\mathcal{C}$ and we send it to its codomain $\mathrm{cod}(f)$. A morphism in $\mathcal{C}$ emanating from $\mathrm{cod}(f)$ is passed back to its composite with f.

Exercise 5.91. Suppose $\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$ is a comonoid.

1. Show that the map $(\mathrm{cod}, \mathrm{com})\colon \mathfrak{c}[i]y^{\mathfrak{c}[i]} \nrightarrow \mathfrak{c}$ from **??** satisfies the conditions necessary for being a cofunctor (identities, codomains, and composites).
2. Find a comonoid structure on the polynomial $p \coloneqq \sum_{i \in \mathfrak{c}(1)} \mathfrak{c}[i]y^{\mathfrak{c}[i]}$ and a cofunctor $p \to \mathfrak{c}$.
3. Is the polynomial map $p \to \mathfrak{c}$ an epimorphism? ◊

Exercise 5.92. Suppose $\mathfrak{c}, \mathfrak{d}, \mathfrak{e}$ are polynomials, each with a comonoid structure, and that $f\colon \mathfrak{c} \to \mathfrak{d}$ and $g\colon \mathfrak{d} \to \mathfrak{e}$ are maps of polynomials.

1. If f and $f \fatsemi g$ are each cofunctors, is g automatically a cofunctor? If so, sketch a proof; if not sketch a counterexample.
2. If g and $f \fatsemi g$ are each cofunctors, is f automatically a cofunctor? If so, sketch a proof; if not sketch a counterexample. ◊

Exercise 5.93. Are cofunctors between posets interesting?

1. Consider the chain poset $[n] \cong \sum_{i=1}^{n} y^i$. How many cofunctors are there from $[m] \to [n]$ for all $m, n \in \{0, 1, 2, 3\}$?
2. What does a cofunctor from y into a poset represent? Is there anything you'd call "asymmetric" about it? ◊

Exercise 5.94.

1. What is the finite set $\{\mathscr{Q}_1, \ldots, \mathscr{Q}_n\}$ of comonoids (defined up to isomorphism) for which the carrier polynomial is $y^2 + y$?
2. For each category $\mathscr{Q}_i$, describe how to imagine a cofunctor $\mathscr{C} \to \mathscr{Q}_i$ from an arbitrary category into it.
3. What cofunctors exist between the various $\mathscr{Q}_i$? ◊

Exercise 5.95. Let S be a set. Describe a way to visualize cofunctors from categories into the state category Sy^S. Feel free to focus on the case where S is a small finite set. Hint: use **??**. ◊

Exercise 5.96.

1. Recall the star-shaped category $y^{\mathsf{n}+1} + \mathsf{n}y$ from **??**. Describe cofunctors into it.
2. Describe cofunctors into Ay for a set A.

3. Describe cofunctors into $(\mathbb{N}, \leq)$.
4. Describe cofunctors into $(\mathbb{N}, \geq)$.
5. Let $y^4 + 2y^2 + y$ denote the commutative square category. List the cofunctors from it to the walking arrow category $y^2 + y$? There should be six or so. ◊

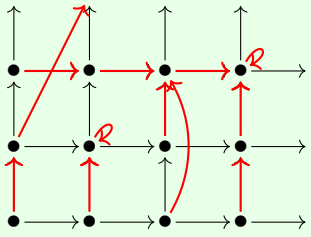
Example 5.97 (Objects aren't representable in $\mathbf{Cat}^\sharp$). For categories and ordinary functors, there is a category $\mathcal{T}$ that *represents objects*, in the sense that functors $\mathcal{T} \to \mathcal{C}$ are the same as objects in $\mathcal{C}$; indeed, take $\mathcal{T} = \boxed{\bullet}$ to be the terminal (one morphism) category.

This does not work for cofunctors, as we'll see in **??**. The comonoid corresponding to $\mathcal{T}$ is y with its unique comonoid structure. Cofunctors $\mathcal{T} \nrightarrow \mathcal{C}$ are somewhat strange beasts: they can be identified with objects $c \in \mathcal{C}$ for which the codomain of every emanating morphism $c \to c'$ is $c' = c$ itself. The reason is the codomain condition (**??**, condition 2).

Exercise 5.98. We saw in **??** that $2y$ has a unique comonoid structure.

1. Show that for any category $\mathcal{T}$, there are $2^{\#\mathrm{Ob}(\mathcal{T})}$-many cofunctors $\mathcal{T} \nrightarrow 2y$.
2. Use the case of $\mathcal{C} := 2y$ to show that if a category $\mathcal{T}$ is going to represent objects as in **??** then $\mathcal{T}$ must have one object.
3. Now use a different $\mathcal{C}$ to show that if a category $\mathcal{T}$ is going to represent objects, it must have more than one object. ◊

Example 5.99 (Policies are co-representable). For a category $\mathcal{C}$, let's say that a *policy in* $\mathcal{C}$ is a choice, for each object $c \in \mathcal{C}$, of an emanating morphism $f : c \to c'$. For example, consider the category $(\mathbb{N}, \leq) \times (\mathbb{N}, \leq)$:



In red we have drawn a policy: every object has been assigned an emanating morphism to another object; there doesn't need to be any rhyme or reason to our choice.

For any category $\mathcal{C}$, the set of trajectories in $\mathcal{C}$ is in bijection with the set of cofunctors

$$\mathcal{C} \nrightarrow n$$

where $n = y^{\mathbb{N}}$ is the monoid of natural numbers under addition.

Exercise 5.100. At the end of **??** we said that a policy on $\mathcal{C}$ can be identified with a cofunctor $F\colon \mathcal{C} \nrightarrow \mathcal{N}$. But at first it appears that F includes more than just a policy: for every object $c \in \mathrm{Ob}(\mathcal{C})$ and natural number $n \in \mathbb{N}$, we have a morphism $F^\sharp_c(n)$ emanating from c. That's infinitely many emanating morphisms per object, whereas a policy seems to include only one emanating morphism per object.

Explain why looks are deceiving in this case: why is a policy on $\mathcal{C}$ the same as a cofunctor $\mathcal{C} \nrightarrow \mathcal{N}$? ◊

We will see later in **??** that the trajectories on a category form a monoid, and that this operation $\mathbf{Cat}^\sharp \to \mathbf{Mon}^{\mathrm{op}}$ is functorial and in fact an adjoint.

Exercise 5.101 (Continuous trajectories). Suppose we say that a continuous policy in $\mathcal{C}$ is a cofunctor $\mathcal{C} \nrightarrow \mathcal{R}$, where $\mathcal{R}$ is the monoid of real numbers under addition, considered as a category with one object.

Describe continuous trajectories in $\mathcal{C}$ using elementary terms, i.e. to someone who doesn't know what a cofunctor is and isn't yet ready to learn. ◊

Exercise 5.102. Let $\mathbb{R}/\mathbb{Z} \cong [0,1)$ be the quotient of $\mathbb{R}$ by the $\mathbb{Z}$-action sending $(r,n) \mapsto r+n$. More down to earth, it's the set of real numbers between 0 and 1, including 0 but not 1.

1. Find a comonoid structure on $(\mathbb{R}/\mathbb{Z})y^{\mathbb{R}}$.
2. Is it a groupoid? ◊

Exercise 5.103.

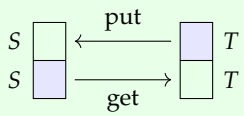
1. If two categories are isomorphic in **Cat**, does that imply they are isomorphic in $\mathbf{Cat}^\sharp$?
2. If so, prove it; if not, give a counterexample.
3. Is it true that for any two categories $\mathcal{C}, \mathcal{D}$, there is a bijection between the set of isomorphisms $\mathcal{C} \xrightarrow{\cong} \mathcal{D}$ in **Cat** and the set of isomorphisms $\mathcal{C} \overset{\cong}{\nrightarrow} \mathcal{D}$ between them in $\mathbf{Cat}^\sharp$?
4. If so, prove it; if not, give a counterexample. ◊

Very well behaved lenses In the functional programming community, there is an important notion of very well-behaved lenses. These turn out to be precisely the cofunctors between state categories. Since state categories Sy^S play an important role in our theory, we take a bit of time to consider cofunctors between them.

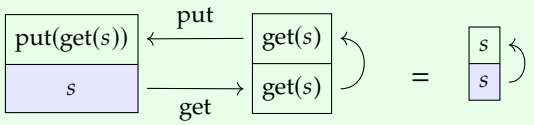
Example 5.104 (Very well-behaved lenses). Recall from **??** that for any set S, we have the "state" category with emanation polynomial Sy^S. What are the comonoid morphisms—

cofunctors—between different state categories?

First, such a comonoid morphism includes a morphism of polynomials $f : Sy^S \to Ty^T$; we'll use the standard terminology of "get" and "put":



Let's apply the unit-homomorphism property (**??**)



It says that put(get(s)) = s. This is typically called the get-put law.

We leave the comultiplication-homomorphism law to **??**, where we will see that it specifies that get and put must satisfy two other properties, called the put-put and the put-get laws.

Exercise 5.105. Complete **??**.

1. Write out the comultiplication law from (**??**) in terms of poly-boxes.
2. What set-theoretic equations are forced by the comultiplication law?
3. Can you see why they might be called put-put and put-get? ◊

Example 5.106 (Very well-behaved lenses are kinda boring). We saw in **??** that a comomonoid homomorphism (cofunctor) $Sy^S \nrightarrow Ty^T$ between state comonoids can be characterized as a pair of functions get: $S \to T$ and put: $S \times T \to S$ satisfying get-put, put-get, and put-put.

In fact, it turns out that this happens if and only if get is a product projection! For example, if the cardinalities $|S|$ and $|T|$ of S and T are finite and $|S|$ is not divisible by $|T|$, then there are no cofunctors $Sy^S \nrightarrow Ty^T$. A stringent condition, no? We'll explore it in **??** below.

Let's explain why cofunctors between state categories are just product projections. A product projection $A \times B \to A$ always has another factor (B); if every cofunctor between state categories is a product projection, what is the other factor†? It turns out that the other factor will be:

$$F \coloneqq \{f : T \to S \mid t \in T,\ \text{get}(f(t)) = t \text{ and } \text{put}(f(t), t) = f(t)\}.$$

In other words we will see that if (get, put) is a comonoid homomorphism then there

is a bijection $S \cong T \times F$ and that get: $S \to T$ is one of the projections. We will see that the converse is true in **??**

So assume (get, put): $Sy^S \nrightarrow Ty^T$ is a comonoid homomorphism, in particular that it satisfies put-get, get-put, and put-put. We obtain a function $\pi\colon S \to T \times F$ given by

$$s \mapsto \big(\text{get}(s), t \mapsto \text{put}(s,t)\big)$$

and it is well-defined since for all $s \in S$ and $t, t' \in T$ we have $\text{get}(\text{put}(s,t)) = t$ by put-get and $\text{put}(\text{put}(s,t),t') = \text{put}(s,t')$ by put-put. We also obtain a function $\pi'\colon T \times F \to S$ given by

$$(t, f) \mapsto f(t).$$

The two functions π, π' are mutually inverse: the roundtrip on S is identity because $\text{put}(s, \text{get}(s)) = s$ by get-put; the roundtrip on $T \times F$ is identity because $\text{get}(f(t)) = t$ and $\text{put}(f(t), t) = f(t)$ by assumption on $f \in F$.

Exercise 5.107. Let S, T, F be sets and suppose given an isomorphism $\alpha\colon S \to T \times F$.

1. Show that there exists a very well behaved lens get: $S \to T$ and put: $S \times T \to S$.
2. Show that there exists a cofunctor between the state category on S and the state category on T.
3. Show that there exists a comonoid homomorphism $Sy^S \to Ty^T$ between the state comonoids. ◊

Exercise 5.108.

1. Suppose $|S| = 3$. How many cofunctors are there $Sy^S \to Sy^S$?
2. Suppose $|S| = 4$ and $|T| = 2$. How many cofunctors are there $Sy^S \nrightarrow Ty^T$? ◊

Exercise 5.109. Let S, T be sets and get: $S \to T$ and put: $S \times T \to S$ the parts of a very well behaved lens, i.e. a cofunctor $Sy^S \to Ty^T$ between state categories. Is it possible that put: $S \times T \to S$ is itself a product projection, i.e. sends $(s,t) \mapsto s$? ◊

When we get to cofree comonoids, we'll obtain a whole new class of cofunctors that are interesting to consider. But for now, we move on to more theory.

Some math about Cat$^\sharp$ We refer to morphisms between polynomial comonoids as cofunctors, again eliding the difference between comonoids in **Poly** and categories.

Proposition 5.110 (Niu)**.** The coproduct of polynomial comonoids agrees with the coproduct of categories. In particular, the initial comonoid is 0.

Proof. We refer the claim about 0 to **??**.

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be categories and $\mathfrak{c}_1, \mathfrak{c}_2$ their emanation polynomials, i.e. the carriers of the corresponding comonoids $\mathscr{C}_1$ and $\mathscr{C}_2$. We first notice that $\mathfrak{c} := \mathfrak{c}_1 + \mathfrak{c}_2$ is the carrier for the comonoid $\mathscr{C}$ corresponding to the sum category $\mathcal{C} := \mathcal{C}_1 + \mathcal{C}_2$. Indeed, the object-set of the sum is given by the sum of the object-sets

$$\mathrm{Ob}(\mathcal{C}_1 + \mathcal{C}_2) \cong \mathrm{Ob}(\mathcal{C}_1) + \mathrm{Ob}(\mathcal{C}_2),$$

and a morphism in $\mathcal{C}_1 + \mathcal{C}_2$ emanating from any such object is just a morphism in whichever of the categories $\mathcal{C}_1$ or $\mathcal{C}_2$ it is from.

It remains to show that $\mathscr{C}$ is the coproduct of $\mathscr{C}_1$ and $\mathscr{C}_2$ in $\mathbf{Cat}^\sharp$. Suppose given a comonoid $\mathscr{D}$ and comonoid homomorphisms (cofunctors) $f_1 : \mathscr{C}_1 \nrightarrow \mathscr{D}$ and $f_2 : \mathscr{C}_2 \nrightarrow \mathscr{D}$. Then for any object of $\mathscr{C}$ we have an associated object $f(c) \in \mathscr{D}$, given either by $f(c) := f_1(c)$ or by $f(c) := f_2(c)$ depending on whether $c \in \mathscr{C}_1$ or $c \in \mathscr{C}_2$. For any morphism m emanating from $f(c)$ we have a morphism $f^\sharp(m)$ emanating from c. It is easy to check that the cofunctor laws hold for f. Uniqueness of f given f_1, f_2 is also straightforward. □

Exercise 5.111.

1. Show that 0 is a comonoid.
2. Show that 0 is initial as a comonoid. ◊

Exercise 5.112. If $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ is a category, show there is an induced category structure on the polynomial $2\mathfrak{c}$. ◊

Exercise 5.113. Check that the terminal comonoid is y. ◊

Proposition 5.114 (Niu)**.** The category $\mathbf{Cat}^\sharp$ has products.

Proposition 5.115. Let $\mathbf{Comon}(\mathbf{Poly})_{\mathrm{rep}}$ be the full subcategory of comonoids (c, ϵ, δ) in **Poly** for which the carrier $c = y^M$ is representable. Then there is an isomorphism of categories

$$\mathbf{Comon}(\mathbf{Poly})_{\mathrm{rep}} \cong \mathbf{Mon}^{\mathrm{op}}$$

where **Mon** is the category of monoids.

Proof. Let $\mathcal{C}$ be a category. It has only one object iff its emanation polynomial $\mathfrak{c}$ has only one position, i.e. $\mathfrak{c} \cong y^M$ for some $M \in \mathbf{Set}$, namely where M is the set of morphisms in $\mathcal{C}$. It remains to show that cofunctors between monoids are dual—opposite—to morphisms between monoids.

A cofunctor $f : y^M \to y^N$ involves a single function $f^\sharp : N \to M$ that must satisfy a law coming from unitality and one coming from composition, as in **??**. The result can now be checked by hand, or seen formally as follows. Each object in the two diagrams (**??**) is representable by **??**. The Yoneda embedding $\mathbf{Set}^{\mathrm{op}} \to \mathbf{Poly}$ is fully faithful, so these two diagrams are equivalent to the unit and composition diagrams for monoid homomorphisms. □

Exercise 5.116. Let $\mathbf{Comon}(\mathbf{Poly})_{\mathrm{lin}}$ be the full subcategory of comonoids (c, ϵ, δ) in **Poly** for which the carrier $c = My$ is linear. Show that there is an isomorphism of categories

$$\mathbf{Comon}(\mathbf{Poly})_{\mathrm{lin}} \cong \mathbf{Set}.$$ ◊

Proposition 5.117. The inclusion of linear comonoids into all comonoids has a left adjoint

$$\mathbf{Comon}(\mathbf{Poly}) \underset{Ay}{\overset{(\mathfrak{c} \triangleleft 1)y}{\underset{\Rightarrow}{\longrightarrow}}} \mathbf{Comon}(\mathbf{Poly})_{\mathrm{lin}}$$

denoted by where they send a comonoid $(\mathfrak{c}, \epsilon, \delta)$ and a linear comonoid Ay.

Proof. We need to show that for any comonoid $(\mathfrak{c}, \epsilon, \delta)$ and set A, we have a natural isomorphism

$$\mathbf{Cat}^\sharp(\mathfrak{c}, Ay). \cong^? \mathbf{Cat}^\sharp((\mathfrak{c} \triangleleft 1)y, Ay)$$

But every morphism in Ay is an identity, so the result follows from the fact that every cofunctor must pass identities back to identities. □

A cofunctor (map of polynomial comonoids) is called *cartesian* if the underlying map $f : \mathfrak{c} \to \mathfrak{d}$ of polynomials is cartesian (i.e. for each position $i \in \mathfrak{c}(1)$, the map $f_i^\sharp : \mathfrak{d}[f_1(i)] \to \mathfrak{c}[i]$ is an isomorphism).

Proposition 5.118. Every cofunctor $f : \mathcal{C} \nrightarrow \mathcal{D}$ factors as a vertical morphism followed by a cartesian morphism

$$\mathcal{C} \overset{\text{vert}}{\nrightarrow} \mathcal{C}' \overset{\text{cart}}{\nrightarrow} \mathcal{D}.$$

Proof. A cofunctor $\mathcal{C} \nrightarrow \mathcal{D}$ is a map of polynomials $\mathfrak{c} \to \mathfrak{d}$ satisfying some properties, and any map of polynomials $f : \mathfrak{c} \to \mathfrak{d}$ can be factored as a vertical morphism followed by a cartesian morphism

$$\mathfrak{c} \overset{g}{\longrightarrow} \mathfrak{c}' \overset{h}{\longrightarrow} \mathfrak{d}.$$

For simplicity, assume $g_1 : \mathfrak{c}(1) \to \mathfrak{c}'(1)$ is identity (rather than merely isomorphism) on positions and similarly that for each $i \in \mathfrak{c}$ the map $h_i^\sharp : \mathfrak{c}'[i] \to \mathfrak{d}[h_1(i)]$ is identity (rather than merely isomorphism) on directions.

It suffices to show that the intermediate object $\mathfrak{c}'$ can be endowed with the structure of a category such that g and h are cofunctors. Given an object $i \in \mathfrak{c}'(1)$, assign its identity to be the identity on $h_1(i) = f(i)$; then both g and h preserve identities because f does. Given an emanating morphism $m \in \mathfrak{c}'[i] = \mathfrak{d}[f(i)]$, assign its codomain to be $\mathrm{cod}(m) := \mathrm{cod}(f_i^\sharp(m))$, and given an emanating morphism $m' \in \mathfrak{c}'[\mathrm{cod}(m)]$, assign the composite $m \fatsemi m'$ in $\mathfrak{c}'$ to be $m \fatsemi m'$ in $\mathfrak{d}$. In **??** we will check that with these definitions, $\mathfrak{c}'$ is a category and both g and h are cofunctors. □

Exercise 5.119. We will complete the proof of **??**, using the same notation.

1. Show that composition is associative and unital in $\mathfrak{c}'$.
2. Show that g preserves codomains.
3. Show that g preserves compositions.
4. Show that h preserves codomains.
5. Show that h preserves compositions. ◊

Proposition 5.120. The wide subcategory of cartesian maps in $\mathbf{Cat}^\sharp$ is isomorphic to the category of wide subcategory of discrete opfibrations in **Cat**.

Proof. Suppose that $\mathcal{C}$ and $\mathcal{D}$ are categories. Both a functor and a cofunctor between them involve a map on objects, say $f\colon \mathrm{Ob}(\mathcal{C}) \to \mathrm{Ob}(\mathcal{D})$. For any object $c \in \mathrm{Ob}(\mathcal{C})$, a functor gives a function, say $f_\sharp\colon \mathcal{C}[c] \to \mathcal{D}[f(c)]$ whereas a cofunctor gives a function $f^\sharp\colon \mathcal{D}[f(c)] \to \mathcal{C}[c]$. The cofunctor is cartesian iff $f^\sharp$ is an iso, and the functor is a discrete opfibration iff $f_\sharp$ is an iso. We thus transform our functor into a cofunctor (or vice versa) by taking the inverse function on morphisms. It is easy to check that this inverse appropriately preserves identities, codomains, and compositions. □

Proposition 5.121. The wide subcategory of vertical maps in $\mathbf{Cat}^\sharp$ is isomorphic to the opposite of the wide subcategory bijective-on-objects maps in **Cat**:

$$\mathbf{Cat}^\sharp_{\mathrm{vert}} \cong (\mathbf{Cat}_{\mathrm{boo}})^{\mathrm{op}}.$$

Proof. Let $\mathcal{C}$ and $\mathcal{D}$ be categories. Given a vertical cofunctor $F\colon \mathcal{C} \nrightarrow \mathcal{D}$, we have a bijection $F_1\colon \mathrm{Ob}(\mathcal{C}) \to \mathrm{Ob}(\mathcal{D})$; let G_1 be its inverse. We define a functor $G\colon \mathcal{D} \to \mathcal{C}$ on objects by G_1 and, for any $f\colon d \to d'$ in $\mathcal{D}$ we define $G(f) := F^\sharp_{G_1(d)}$. It has the correct codomain: $\mathrm{cod}(G(f)) = G_1(F_1(\mathrm{cod}(G(f))) = G_1(\mathrm{cod}\, f)$. And it sends identities and compositions to identities and compositions by the laws of cofunctors.

The construction of a vertical cofunctor from a bijective-on-objects functor is analogous, and it is easy to check that the two constructions are inverses. □

Exercise 5.122. Let S be a set and consider the state category $\mathcal{S} := (Sy^S, \epsilon, \delta)$. Use **??** to show that categories $\mathcal{C}$ equipped with a vertical cofunctor $\mathcal{S} \to \mathcal{C}$ can be identified with categories whose set of objects is S. ◊

Exercise 5.123. Consider the categories $\mathcal{C} := \boxed{\bullet \rightrightarrows \bullet}$ and $\mathcal{D} := \boxed{\bullet \to \bullet}$. There is a unique bijective-on-objects (boo) functor $F\colon \mathcal{C} \to \mathcal{D}$ and two boo functors $G_1, G_2\colon \mathcal{D} \to \mathcal{C}$.

1. Write down the morphism $\mathfrak{d} \to \mathfrak{c}$ of emanation polynomials underlying F.
2. Write down the morphism $\mathfrak{c} \to \mathfrak{d}$ of emanation polynomials underlying either G_1 or G_2. ◊

Dirichlet monoidal product on $\mathbf{Cat}^\sharp$. The usual product of categories gives a monoidal operation on comonoids too, even though it is not a product in $\mathbf{Cat}^\sharp$. The carrier polynomial of the product is the $\otimes$-product of the carrier polynomials.

Proposition 5.124. The Dirichlet monoidal product $(y, \otimes)$ on **Poly** extends to a monoidal structure $(y, \otimes)$ on $\mathbf{Cat}^\sharp$, such that the functor $U\colon \mathbf{Cat}^\sharp \to \mathbf{Poly}$ is strong monoidal with respect to $\otimes$. The Dirichlet product of two categories is their product in **Cat**.

Proof. Let $\mathcal{C}, \mathcal{D} \in \mathbf{Cat}^\sharp$ be categories with emanation polynomials $\mathfrak{c}, \mathfrak{d} \in \mathbf{Poly}$. The emanation polynomial of $\mathcal{C} \otimes \mathcal{D}$ is defined to be $\mathfrak{c} \otimes \mathfrak{d}$. A position in it is a pair (c, d) of objects, one from $\mathcal{C}$ and one from $\mathcal{D}$; a direction there is a pair (f, g) of a morphism emanating from c and one emanating from d.

We define $\epsilon_{\mathcal{C}\otimes\mathcal{D}}\colon \mathfrak{c} \otimes \mathfrak{d} \to y$ as

$$\mathfrak{c} \otimes \mathfrak{d} \xrightarrow{\epsilon_{\mathfrak{C}} \otimes \epsilon_{\mathfrak{D}}} y \otimes y \cong y.$$

This says that the identity at (c, d) is the pair of identities.

We define $\delta_{\mathcal{C}\otimes\mathcal{D}}\colon (\mathfrak{c} \otimes \mathfrak{d}) \to (\mathfrak{c} \otimes \mathfrak{d}) \triangleleft (\mathfrak{c} \otimes \mathfrak{d})$ using the duoidal property:

$$\mathfrak{c} \otimes \mathfrak{d} \xrightarrow{\delta_{\mathfrak{c}} \otimes \delta_{\mathfrak{d}}} (\mathfrak{c} \triangleleft \mathfrak{c}) \otimes (\mathfrak{d} \triangleleft \mathfrak{d}) \to (\mathfrak{c} \otimes \mathfrak{d}) \triangleleft (\mathfrak{c} \otimes \mathfrak{d}).$$

One can check that this says that codomains and composition are defined coordinatewise, and that $(\mathfrak{c} \otimes \mathfrak{d}, \epsilon_{\mathfrak{c}\otimes\mathfrak{d}}, \delta_{\mathfrak{c}\otimes\mathfrak{d}})$ forms a comonoid. One can also check that this is functorial in $\mathcal{C}, \mathcal{D} \in \mathbf{Cat}^\sharp$. See **??**. □

Exercise 5.125. We complete the proof of **??**.

1. Show that $(\mathfrak{c} \otimes \mathfrak{d}, \epsilon_{\mathfrak{c}\otimes\mathfrak{d}}, \delta_{\mathfrak{c}\otimes\mathfrak{d}})$, as described in **??**, forms a comonoid.
2. Check that the construction $(\mathcal{C}, \mathcal{D}) \mapsto \mathcal{C} \otimes \mathcal{D}$ is functorial in $\mathcal{C}, \mathcal{D} \in \mathbf{Cat}^\sharp$. ◊

5.3 Cofree comonoids

5.3.1 Introduction: Cofree comonoids and dynamical systems

We can now return to dynamical systems. Recall that if $p \in \mathbf{Poly}$ is a polynomial, then a dynamical system with interface p consists of a set S and a map of polynomials $f : Sy^S \to p$. We think of positions in p kind of like outputs—others can observe your position—but also as determining the set of inputs—directions—that could be input next.

The way this map f leads to something that seems to "go on repeatedly" is that $\mathfrak{s} \coloneqq Sy^S$ is a comonoid, so we have maps $\mathfrak{s} \xrightarrow{\delta} \mathfrak{s}^{\triangleleft n} \xrightarrow{f^{\triangleleft n}} p^{\triangleleft n}$ for all n. This says that given any initial position in S, we automatically get a position in p, and for every direction there, another position in p, and for every direction there, another position in p, and so on n times. This is the dynamics.

So the above all works because we have a polynomial map $Sy^S \to p$, where Sy^S is the underlying polynomial of a polynomial comonad. Since "underlying polynomial" is a functor $U : \mathbf{Cat}^\sharp \to \mathbf{Poly}$, a seasoned category theorist might be tempted to ask, is there an adjunction

$$\mathbf{Poly}(U\mathscr{C}, p) \cong \mathbf{Cat}^\sharp(\mathscr{C}, \mathscr{T}_p),$$

for some functor $\mathscr{T}$: $\mathbf{Poly} \to \mathbf{Cat}^\sharp$? In fact, there is; we refer to $\mathscr{T}_p$ as the *cofree comonoid* on p, or more descriptively, the *category of p-trees.*

Cofree comonoids in **Poly** are beautiful objects, both in their visualizable structure as a category and in the metaphors we can make about them. They allow us to replace the interface of a dynamical system with a category and get access to a rich theory that exists there.

Theorem 5.126 (Cofree comonoid)**.** The forgetful functor $\mathbf{Comon}(\mathbf{Poly}) \to \mathbf{Poly}$ has a right adjoint

$$\mathbf{Cat}^\sharp \underset{\mathscr{T}_p}{\overset{\mathfrak{c}}{\underset{\Rightarrow}{\rightleftarrows}}} \mathbf{Poly}$$

where the functors have been named by where they send $(\mathfrak{c}, \epsilon, \delta) \in \mathbf{Cat}^\sharp$ and $p \in \mathbf{Poly}$ respectively.

This will be proved as **??**.

5.3.2 Cofree comonoids as trees

Definition 5.127 (Cofree comonoid)**.** Let $p \in \mathbf{Poly}$ be a polynomial. The comonoid $\mathscr{T}_p = (\mathsf{tree}_p, \mathrm{root}, \mathrm{focus})$ as in **??** is called the *cofree comonoid on p* or informally the category of *(possibly infinite) p-trees*.

An object $t \in \mathsf{tree}_p(1)$ is a called a *(possibly infinite) tree in p*. Given such an object t

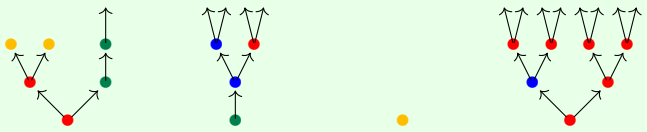
in the category, an emanating morphism $n \in \mathsf{tree}_p[t]$ is called a *path from root*.

The terminology of **??** is alluding to a specific way we like to imagine the comonoid $\mathscr{T}_p = (\mathsf{tree}_p, \mathrm{root}, \mathrm{focus})$, namely in terms of trees. To every polynomial p, we will associate a new polynomial tree_p whose positions are (possibly infinite) p-trees. To choose such tree we first choose its root to be some position $i \in p(1)$. Then for every direction $d \in p[i]$ there, we choose another position, and for every direction from each of those we choose another position, and so on indefinitely.

So a position in tree_p is one of these trees. Such a tree may end, namely if every one of the top-level positions have no directions, but often it will not end. Given such a tree, say t, a direction $d \in \mathsf{tree}_p[t]$ there is simply a path from the root to some node in the tree.

We'll explain the counit root and the comultiplication focus after going through an example.

Example 5.128. Let $p := \{\bullet, \bullet\}y^2 + \bullet y + \bullet$. Here are four trees in p:



(5.129)

They all represent elements of $p^{\triangleleft 3}$, but only the third one—the single yellow dot—would count as an element of tree_p.

Indeed, in **??**, when we speak of a (possibly infinite) tree in p, we mean at tree for which each node is a position in p with each of its emanating directions filled by another position in p, and so on. Since three of the four trees shown in (**??**) have open leaves—arrows emanating from the top—these trees are not elements of tree_p. However, each of them could be extended to an actual element of tree_p by continually filling in each open leaf with another position of p.

Let's imagine some actual elements of tree_p:

- The binary tree that's "all red all the time".
- The binary tree where odd layers are red and even layers are blue.
- The tree where all the nodes are red, except for the right-most branch, which is always green.[a]
- Any finite tree, where every branch terminates in a yellow dot.
- Completely random: for the root, randomly choose either red, blue, green, or yellow, and at every leaf, loop back to the beginning, i.e. randomly choose either red, blue, green, or yellow, etc.

These are the positions in the polynomial tree_p that underlies the cofree comonoid on p. There are uncountably many positions in $\mathscr{T}_p$, at least for this particular p—in fact, even $\mathscr{T}_{2y}$ has $2^{\mathbb{N}}$-many positions—but only finitely many can be described in any finite

language like English. Thus most of the elements of $\mathscr{T}_p$ cannot be described.

[a] Note that branches are actually unordered, so it's technically wrong to think of it as a line of green up the *right side*. Instead, it's just a line of green going up the tree forever.

Exercise 5.130.

1. Interpret each of the five tree examples imagined in **??** by drawing three or four layers (your choice) of it.

For each of the following polynomials p, describe the set of trees (positions) in tree_p.

2. $p = 1$. (What is the set tree_p of p-trees?)
3. $p = 2$.
4. $p = y$.
5. $p = y^2$.
6. $p = 2y$.
7. $p = y + 1$.
8. $p = By^A$ for some sets $A, B \in \mathbf{Set}$. ◊

Now that we've explained the underlying polynomial tree_p of the cofree comonoid $\mathscr{T}_p = (\mathsf{tree}_p, \mathsf{root}, \mathsf{focus})$, we just need to explain how identities, codomains, and composition work, i.e. we just need to give the counit map $\mathsf{root}\colon \mathsf{tree}_p \to y$ and the comultiplication map $\mathsf{focus}\colon \mathsf{tree}_p \to \mathsf{tree}_p \triangleleft \mathsf{tree}_p$.

Again, the objects in the category $\mathscr{T}_p$ are p-trees, and a morphism emanating from such a tree t is a path from its root r to some node. The map root, applied to t, returns t's root r, or more precisely the path from r to itself. The map focus, applied to t, first needs to give a codomain (tree) to every path from the root to some other node n. It is just the subtree of t whose root is n: the tree of all nodes starting at n. Now, given a path from the root of that tree (namely n) to another node, say n', we need to give a path from r to n'; we take it to be the composite of the path $r \to n$ and the path $n \to n'$.

Exercise 5.131. Let $p \coloneqq \{\bullet, \bullet\}y^2 + \bullet y + \bullet$ as in **??**.

1. Choose an object $t \in \mathsf{tree}_p$, i.e. a tree in p, and draw a finite approximation of it (say four layers).
2. What is the identity morphism at t?
3. Choose a nonidentity morphism f emanating from t and draw it.
4. What is the codomain of f? Draw a finite approximation of it.
5. Choose a morphism emanating from the codomain of f and draw it.
6. What is the composite of your two morphisms? Draw it on t. ◊

Example 5.132. Let's take $p \coloneqq 1$. An element in $\mathsf{tree}_p(1)$ is given by choosing an element $i \in p(1)$ and filling each of its direction $p[i]$ with another element of $p(1)$, and so on. But there is only one element of $p(1)$ and it has no directions. So tree_1 has only one position,

and the only emanating morphism there is the identity. In other words, $\mathsf{tree}_1 = y$.

Since y has a unique comonoid structure, we've described the cofree comonoid $\mathscr{T}_{\!(}1)$. It is a single tree consisting of a single node, and the only outgoing morphism is the identity on that node.

Exercise 5.133. Let A be a set.

1. What is tree_A?
2. How is it given the structure of a category? ◊

Example 5.134. Let $p := y$. An element in $\mathsf{tree}_p(1)$ is given by choosing an element $i \in p(1)$ and filling each of its direction $p[i]$ with another element of $p(1)$, and so on. There is only one way to do this, i.e. there is only one such tree, namely $t := \boxed{\bullet \to \bullet \to \bullet \to \cdots}$.

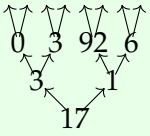
So tree_p has a single position, namely t. That position has an emanating morphism for each path out of the root, so it has $\mathbb{N}$-many emanating morphisms: one for every length. Hence $\mathsf{tree}_y = y^{\mathbb{N}}$.

Of course the codomain of each morphism emanating from t is again t: that's the only object. The composite of two paths, one of length m and one of length n is $m + n$. Hence we see that the category $\mathscr{T}_{\!(}y)$ is the monoid $(\mathbb{N}, 0, +)$ considered as a category with one object.

Exercise 5.135. Let A be a set.

1. What is tree_{Ay}?
2. How is it given the structure of a category? ◊

Example 5.136. Let $p := \mathbb{N}y^2$. An element of tree_p might start like this:



Any element of tree_p goes on forever: it's an infinite binary tree. At each node it has a choice of some natural number, since $\mathbb{N} = p(1)$ is the set of positions in p.

So such trees are the objects of the category $\mathscr{T}_p = (\mathsf{tree}_p, \mathsf{root}, \mathsf{focus})$. A morphism emanating from a tree t is a path from its root to another node, which is an element of $\mathsf{List}(2)$, i.e. a finite list of choices in 2, which you can think of as a finite sequence of left/right choices. The codomain is whatever tree this path ends up on.

So the emanation polynomial of $\mathscr{T}_p$ is

$$\mathsf{tree}_p \cong \mathbb{N}^{\mathsf{List}(2)} y^{\mathsf{List}(2)}$$

with identities given by the empty list. An object $t \in \mathsf{tree}_p(1)$ is a function $t : \mathsf{List}(2) \to \mathbb{N}$, a way to put a natural number at every node of the infinite binary tree. An emanating morphism $\ell \in \mathsf{List}(2)$ is just a path from the root to another node, and its codomain is the other node. Formally it is the function $t' : \mathsf{List}(2) \to \mathbb{N}$ given by $t'(\ell') := t(\ell : \ell')$, where $\ell : \ell'$ is the concatenation of these lists. Composition of morphisms is also given by concatenation of the corresponding lists.

Exercise 5.137. Let $p := By^A$ for sets $A, B \in \mathbf{Set}$.

1. Describe the objects of the cofree category $\mathscr{T}_p$.
2. For a given such object, describe the emanating morphisms.
3. Describe how to take the codomain of a morphism. ◊

Exercise 5.138. Let $p := y + 1$.

1. Describe the objects of the cofree category $\mathscr{T}_p$.
2. For a given such object, describe the emanating morphisms.
3. Describe how to take the codomain of a morphism. ◊

Exercise 5.139. Let $p := \{a, b, c, \ldots, z, \sqcup\}y + 1$.

1. Describe the objects of the cofree category $\mathscr{T}_p$, and draw one.
2. For a given such object, describe the emanating morphisms.
3. Describe how to take the codomain of a morphism. ◊

Decision trees. When you talk about your future, what exactly might you be talking about? In some sense you can make choices that change what will happen to you, but in another sense it's as though for each such choice there is something beyond your control that makes a new situation for you. You're constantly in the position of needing to make a choice, but its results are beyond your control.

This is very much how positions $t \in \mathscr{T}_p$ look. Such a position is a decision tree: at each stage (node), you have an element $i \in p(1)$, which we've been calling a decision. It has $p[i]$-many options, each of which, say $d \in p[i]$ results in a new node $\mathrm{cod}(d)$ of the tree.

So a position t is like a future: it is a current decision, and for every option there, a new decision tree. It's all the decisions you could possibly make, and for each actual choice, it's a new future. A direction at t is just a choice of finite path through the tree: a sequence of choices.

Exercise 5.140. If someone says that they understand a future to be a decision tree $t \in \mathscr{T}_p$, explain in your own words how they're thinking about the term "future". How does it agree or disagree with your own intuition about what a "future" is? ◊

5.3.3 Formal construction of $\mathscr{T}_p$

We will sketch how one can formally construct $\mathscr{T}_p$ from p. The first step is called copointing, and it's pretty easy: just multiply p by y. It adds a kind of "default" direction to each position in p.

Copointing.

Definition 5.141 (Copointed polynomial)**.** A *copointed polynomial* is a pair (p, ϵ), where $p \in \mathbf{Poly}$ is a polynomial and $\epsilon : p \to y$ is a morphism in **Poly**.

A *morphism* of copointed polynomials $f : (p, \epsilon) \to (p', \epsilon')$ is a morphism $f : p \to p'$ such that $\epsilon = f \fatsemi \epsilon'$.

Comonoids in **Poly** are triples (p, ϵ, δ), with (p, ϵ) a copointed polynomial, so there are forgetful functors

$$\mathbf{Comon}(\mathbf{Poly}) \to \mathbf{Copnt}(\mathbf{Poly}) \to \mathbf{Poly}.$$

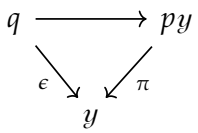
We want to find the right adjoint to the composite—that's the functor $\mathscr{T}_- : \mathbf{Poly} \to \mathbf{Comon}(\mathbf{Poly})$—and we will obtain it in two steps.

Proposition 5.142. For any polynomial p, the polynomial py is naturally copointed by the projection to y, and the functor sending $p \mapsto py$ is right adjoint to the forgetful functor

$$\mathbf{Copnt}(\mathbf{Poly}) \underset{q}{\overset{py}{\underset{\longleftarrow}{\overset{\longrightarrow}{\Rightarrow}}}} \mathbf{Poly} \ ,$$

where the functors are named by where they send $p \in \mathbf{Poly}$ and $(q, \epsilon) \in \mathbf{Copnt}(\mathbf{Poly})$.

Proof. Clearly the product $py \cong p \times y$ is copointed by the projection map, call it $\pi : py \to y$, and the map $p \mapsto py$ is functorial in p. For any copointed polynomial $q \xrightarrow{\epsilon} y$, there is an obvious bijection between morphisms of polynomials $q \to p$ and commutative triangles



natural in both q and p. This completes the proof. □

The reader might not remember any sort of copointing showing up in the tree description of $\mathcal{T}_p = (\mathsf{tree}_p, \mathsf{root}, \mathsf{focus})$. Indeed, it was hidden in the fact that we allowed for trivial paths in the tree (e.g. the path from the root to itself). But we'll get to that.

The copointing $p \mapsto py$ just adds an extra direction to each position; we can denote this extra direction with an =, as we did in **??**. So for example if $p = y^3 + y$, drawn as left, then $py \cong y^4 + y^2$ can be drawn as right:

p $\qquad$ py

It just adds a default direction to each position. A copointed map from (q, ϵ) to (py, π) must pass the default direction back to the default direction in q, but leaves the other directions in p to go wherever they want to.

Example 5.143 (Slowing down dynamical systems). Given a dynamical system $f : Sy^S \to p$, we automatically get a map $Sy^S \to py$ to the cofree pointing. We called this "adding a pause button" in Example 4.130. Thus we can take any dynamical system and replace it with one whose interface is copointed.

We can use a copointed interface to slow down a dynamical system, in a kind of inverse to how we sped up dynamical systems in (**??**). There we took a dynamical system f with interface p and produced one and produced one with interface $p^{\triangleleft k}$. Here we will take one with interface $p^{\triangleleft k}$ and produce one with interface p.

To do this, we need p to be copointed, i.e. we need to have in hand a map $\epsilon : p \to y$, and as we saw above that we can always assume that. Now for any $k \in \mathbb{N}$ we have k-many maps $p^{\triangleleft k} \to p$. For example, if $k = 3$, we have

$$\{\epsilon \triangleleft \epsilon \triangleleft p, \epsilon \triangleleft p \triangleleft \epsilon, p \triangleleft \epsilon \triangleleft \epsilon\} \subseteq \mathbf{Poly}(p^{\triangleleft 3}, p)$$

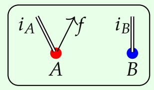
So given a dynamical system $Sy^S \to p^{\triangleleft k}$, which outputs as its position a whole k-fold strategy at one time and which takes as input sequences of k-many inputs, we can feed it one input and $k-1$ pauses. This is what you get when you compose $Sy^S \to p^{\triangleleft k} \to p$.

Given a dynamical system $f : Sy^S \to p$, where p is copointed and f preserves the copoint, we could speed it up as before to get $Sy^S \to p^{\triangleleft k}$ and then slow it down to get $Sy^S \to p$, and we get back f. So slowing down is a retract of speeding up in this sense.

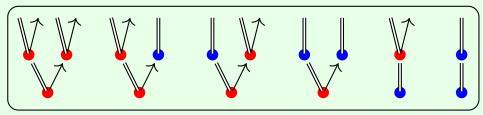
Constructing the cofree comonoid. It remains to show that we can functorially take any copointed polynomial (q, ϵ) and return a comonoid, and that this construction is right adjoint to the forgetful functor. From the description in **??**, we know the cofree comonoid is supposed to have something to do with infinite trees. And we know that the set of height-n p-trees is given by $p^{\triangleleft n}$. So we might think we can somehow take a limit of these height-n trees for various n.

The problem is there's no obvious maps between $p^{\triangleleft n}$ and $p^{\triangleleft n+1}$. Luckily, the copointing fixes that problem. Given $\epsilon\colon q \to y$, we have two maps $q \triangleleft q \rightrightarrows q$, namely $q \triangleleft \epsilon$ and $\epsilon \triangleleft q$.

Example 5.144. Suppose $q = \{A\}y^{\{i_A, f\}} + \{B\}y^{i_B}$ with copointing ϵ selecting i_A and i_B:

$q :=$ 

Then $q \triangleleft q$ looks as follows

$q \triangleleft q =$ 

How can we picture the maps $(q \triangleleft \epsilon), (\epsilon \triangleleft q)\colon q \triangleleft q \to q$?

The map $q \triangleleft \epsilon$ takes each position of $q \triangleleft q$ to whatever is on the bottom layer: it takes the first four to A and the last two to B. It passes back directions using the defaults (i_A and i_B) on the top layer.

The map $\epsilon \triangleleft q$ uses the defaults on the bottom layer instead. Every position in $q \triangleleft q$ has a default direction, and the corolla sitting there in the top layer is where $\epsilon \triangleleft q$ sends it, with identity on directions.

Indeed, for every n, there are $(n+1)$-many morphisms $q^{\triangleleft n+1} \to q^{\triangleleft n}$, so we have a diagram

$$y \xleftarrow{\ \epsilon\ } q \;\underset{q \triangleleft \epsilon}{\overset{\epsilon \triangleleft q}{\leftleftarrows}}\; q \triangleleft q \;\xleftarrow[\ q\triangleleft q\triangleleft\epsilon\ ]{\ \epsilon\triangleleft q\triangleleft q\ ,\ q\triangleleft\epsilon\triangleleft q\ }\; q^{\triangleleft 3} \;\leftleftarrows\!\leftleftarrows\; \cdots \tag{5.145}$$

The cofree comonoid is given by the limit of this diagram.

Let's denote the shape of this diagram by Δ_+: its objects are finite ordered sets—including the empty set—and its morphisms are order-preserving injections. For any copointed polynomial $q \xrightarrow{\epsilon} y$, we get a diagram $Q\colon \Delta_+ \to \mathbf{Poly}$ as above, and this is functorial in q.

Theorem 5.146. For any copointed polynomial $q \xrightarrow{\epsilon} y$, let $\bar{q}$ denote the limit of $Q\colon \Delta_+ \to \mathbf{Poly}$. It naturally has the structure of a comonoid $(\bar{q}, \epsilon, \delta)$, and this construction is right adjoint to the forgetful functor

$$\mathbf{Comon}(\mathbf{Poly}) \underset{\bar{q}}{\overset{U}{\rightleftarrows}} \mathbf{Copnt}(\mathbf{Poly}) \ .$$

Proof sketch. We first give $\bar{q}$ the structure of a comonoid. Since $\bar{q}$ is the limit of (**??**), the inclusion the inclusion $\{0\} \to \Delta_+$ induces a projection map $\bar{q} \to y$, which we again call

ϵ. Since $\triangleleft$ commutes with connected limits in both variables and Δ_+ is connected, we have that $\bar{q} \triangleleft \bar{q}$ is the limit of the following $\Delta_+ \times \Delta_+$-shaped diagram:

$$\bar{q} \triangleleft \bar{q} \cong \lim \left(\begin{array}{ccccccccc} q^{\triangleleft(0+0)} & \longleftarrow & q^{\triangleleft(0+1)} & \Lleftarrow & q^{\triangleleft(0+2)} & \Lleftarrow & q^{\triangleleft(0+3)} & \Lleftarrow & \cdots \\ \uparrow & & \uparrow & & \uparrow & & \uparrow & & \\ q^{\triangleleft(1+0)} & \longleftarrow & q^{\triangleleft(1+1)} & \Lleftarrow & q^{\triangleleft(1+2)} & \Lleftarrow & q^{\triangleleft(1+3)} & \Lleftarrow & \cdots \\ \upuparrows & & \upuparrows & & \upuparrows & & \upuparrows & & \\ q^{\triangleleft(2+0)} & \longleftarrow & q^{\triangleleft(2+1)} & \Lleftarrow & q^{\triangleleft(2+2)} & \Lleftarrow & q^{\triangleleft(2+3)} & \Lleftarrow & \cdots \\ \Uparrow & & \Uparrow & & \Uparrow & & \Uparrow & & \\ q^{\triangleleft(3+0)} & \longleftarrow & q^{\triangleleft(3+1)} & \Lleftarrow & q^{\triangleleft(3+2)} & \Lleftarrow & q^{\triangleleft(3+3)} & \Lleftarrow & \cdots \\ \Uparrow & & \Uparrow & & \Uparrow & & \Uparrow & & \\ \vdots & & \vdots & & \vdots & & \vdots & & \ddots \end{array} \right)$$

There is a commutative diagram in **Cat**

$$\begin{array}{ccc} \Delta_+ \times \Delta_+ & \xrightarrow{+} & \Delta_+ \\ & {\scriptstyle (m_1,m_2)\mapsto q^{\triangleleft(m_1+m_2)}} \searrow \quad \swarrow {\scriptstyle n\mapsto q^{\triangleleft n}} & \\ & \textbf{Poly} & \end{array} \tag{5.147}$$

which induces a map (in the opposite direction) between their limits $\delta\colon \bar{q} \to \bar{q}\triangleleft\bar{q}$, which we take to be the comultiplication. Appending (**??**) with the inclusion $\{0\} \times \Delta_+ \to \Delta_+ \times \Delta_+$, etc., it is easy to see that $(\bar{q}, \epsilon, \delta)$ satisfies the axioms of a comonoid.

We sketch the proof that this construction is right adjoint to the forgetful functor. For any copointed polynomial (q, ϵ), there is a counit map $\bar{q} \to q$, given by the obvious projection of the limit (**??**). Given a comonoid $(\mathfrak{c}, \epsilon, \delta)$, there is a morphism $\mathfrak{c} \to \bar{\mathfrak{c}}$ induced by the maps $\delta^{n-1}\colon \mathfrak{c} \to \mathfrak{c}^{\triangleleft n}$ from **??**. It is easy to check that these commute with the ϵ's in the diagram. To see that $\mathfrak{c} \to \bar{\mathfrak{c}}$ extends to a morphism of comonoids amounts to checking that the diagram

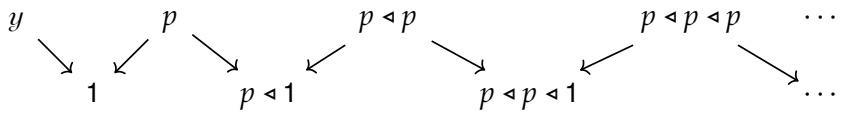
$$\begin{array}{ccc} \mathfrak{c} & \xrightarrow{\delta^{m+n-1}} & \mathfrak{c}^{\triangleleft(m+n)} \\ {\scriptstyle\delta}\downarrow & & \| \\ \mathfrak{c} \triangleleft \mathfrak{c} & \xrightarrow[\delta^{m-1}\triangleleft\delta^{n-1}]{} & \mathfrak{c}^{\triangleleft m} \triangleleft \mathfrak{c}^{\triangleleft n} \end{array}$$

commutes for any $m, n \in \mathbb{N}$. Both triangle equations are straightforward. □

Remark 5.148. The construction of the cofree comonoid from a copointed endofunctor in the proof of **??** is fairly standard; see [**lack2010note**]. Nelson Niu has also constructed the cofree comonoid on a polynomial using a different limit diagram

$$\begin{array}{ccccccccc} y & & p & & p \triangleleft p & & p \triangleleft p \triangleleft p & & \cdots \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\ 1 & \xleftarrow[!]{} & p \triangleleft 1 & \xleftarrow[p\triangleleft !]{} & p \triangleleft p \triangleleft 1 & \xleftarrow[p\triangleleft p\triangleleft !]{} & p \triangleleft p \triangleleft p \triangleleft 1 & \longleftarrow & \cdots \end{array}$$

in terms of the original polynomial p, rather than from its copointing py; that is, this construction is right adjoint to **Comon**(**Poly**) $\to$ **Poly**. One could also construct this right adjoint using the following limit, again applied to the original polynomial p:

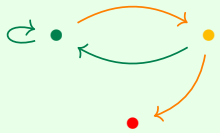


5.3.4 Consequences of adjointness

Recall from Definition 4.127 that a dependent system (or generalized Moore machine) is a map of polynomials $f : Sy^S \to p$. Here S is called the set of states and p is the interface.

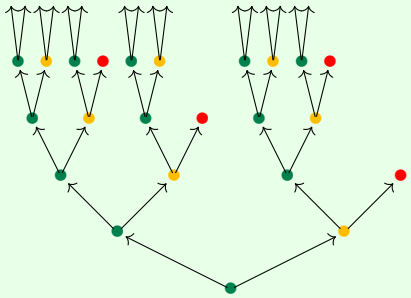
But now we know that Sy^S is secretly the underlying polynomial of a comonoid. This means that f has a mate, i.e. a corresponding cofunctor $Sy^S \nrightarrow \mathcal{T}_p$ to the category of p-trees. How does that work?

Example 5.149. Let $S := \{\bullet, \bullet, \bullet\}$ and $p := y^2 + y$, and consider the dynamical system $f : Sy^S \to p$ from Exercise 4.129, depicted here again for your convenience:



The polynomial map f is supposed to induce a cofunctor $F : Sy^S \nrightarrow \mathcal{T}_p$ from the state category on S to the category of p-trees. Thus to each state $s \in S$, we need to associate a tree; which should it be?

The answer is that we associate to each state its tree of possible futures. For example, to the green state we associate the tree



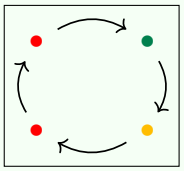
Let's call it t. We will say how F acts on the other two states in **??**, but before doing so, let's say how F passes back directions. A direction $d \in \mathrm{tree}_p[t]$ is just a finite path, say left-left-right, coming out of the root. Each arrow $d \in p[\pi(t)]$ out of the root is passed

back to a new state $\mathrm{cod}(f^\sharp(d)) \in S$. This new state maps to the codomain of d, so we can repeat this procedure n times.

Exercise 5.150. Let's finish off **??**.

1. To what tree does the cofunctor F send the yellow state?
2. To what tree does the cofunctor F send the red state? ◊

Exercise 5.151. Let $S := 4$ and $p := \{\bullet, \bullet, \bullet\}y$, and let $f : Sy^S \to p$ be the dynamical system depicted as follows:



Let $s \in S$ be the state for which $f_1(s) = \bullet$.

If $F : Sy^S \nrightarrow \mathscr{T}_p$ is the associated cofunctor, what is the tree $F_1(s)$? ◊

Exercise 5.152. Suppose S is a set, p is a polynomial, $f : Sy^S \to p$ is a dynamical system, and $F : Sy^S \nrightarrow \mathscr{T}_p$ is the associated cofunctor. To every state $s \in S$, we get a tree $F(s) \in \mathrm{tree}_p$. What common feature do all such trees have when S is a finite set? What do these trees look like? ◊

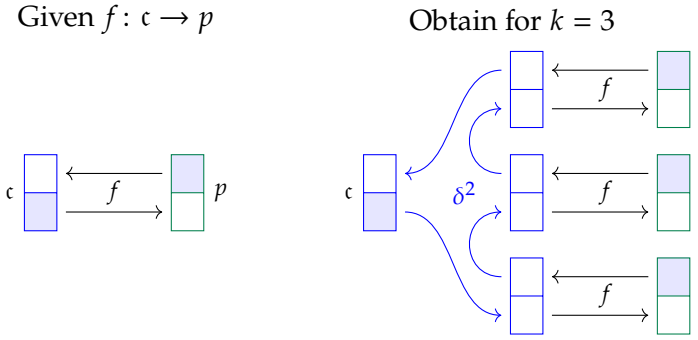
Exercise 5.153. In **??** we considered a dynamical $\mathbb{N}y^{\mathbb{N}} \to \mathbb{N}y$ that acts like a stop watch, ticking up for ever.

1. Obtain a map of polynomials $\mathbb{N}y^{\mathbb{N}} \to y$ from adjointness of the cofree construction.
2. Obtain a map of polynomials $\mathbb{N}y^{\mathbb{N}} \to \mathbb{N}$.
3. Does the pairing of the two maps you found have the stopwatch behavior we were looking for? ◊

The way the adjunction works is as follows. Suppose that $\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$ is a comonoid and p is a polynomial. Then given a map of polynomials $f : \mathfrak{c} \to p$, we can form a map $\mathfrak{c} \to p^{\triangleleft k}$ for any $k \in \mathbb{N}$ as follows. First use the comonoid structure to obtain $\delta^{k-1} : \mathfrak{c} \to \mathfrak{c}^{\triangleleft k}$ as in **??**. Second we use the monoidality of $\triangleleft$ to get $f^{\triangleleft k} : \mathfrak{c}^{\triangleleft k} \to p^{\triangleleft k}$. Composing we obtain

$$\mathfrak{c} \xrightarrow{\delta^{k-1}} \mathfrak{c}^{\triangleleft k} \xrightarrow{f^{\triangleleft k}} p^{\triangleleft k}$$

This is a generalization of what we called "speeding up dynamical systems" in **??**. In terms of poly-boxes it looks like this for $k = 3$:

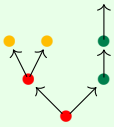


(5.154)

So for any $k \in \mathbb{N}$, we have a map from $\mathfrak{c}$ to p-trees of height k, and we put them together to obtain the map $\mathcal{C} \to \mathscr{T}_p$. The fact that we have an adjunction $\mathbf{Cat}^\sharp \rightleftarrows \mathbf{Poly}$ means that there is nothing more or less in a cofunctor $\mathcal{C} \to \mathscr{T}_p$ than the map $\mathfrak{c} \to p$ of polynomials.

It is also worthwhile understanding the unit and counit of this cofree adjunction. The counit is a map of polynomials $\mathsf{tree}_p \to p$ natural in $p \in \mathbf{Poly}$. On positions, it sends a tree t to its root corolla. Given a direction in the root corolla, we obtain a direction in t, i.e. a path (of length 1) emanating from the root.

Example 5.155. Let's consider $p = \{\bullet, \bullet\}y^2 + \bullet y + \bullet$ as in **??**. Given a p-tree t that starts like this:



its root corolla is simply

So the map $\mathsf{tree}_p \to p$ sends t to $\bullet$. It passes each of the two directions emanating from $\bullet$ to the corresponding direction emanating from the root of t.

Exercise 5.156. Let $p := y^2 + 1$.

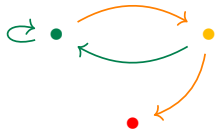
1. Draw a p-tree, i.e. an element $t \in \mathsf{tree}_p(1)$.
2. What is the root corolla of t?
3. To what position does the map $\pi : \mathsf{tree}_p \to p$ send t?
4. For each direction in $p[\pi(t)]$, show the direction in $\mathsf{tree}_p[t]$ that it is passed back to by $\pi^\sharp$. ◊

Given a comonoid $\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$, the unit cofunctor $\mathcal{C} \to \mathscr{T}_\mathfrak{c}$ sends each position (object $c \in \mathcal{C}$) to the tree that puts at the root the set of all morphisms emanating from c, to each

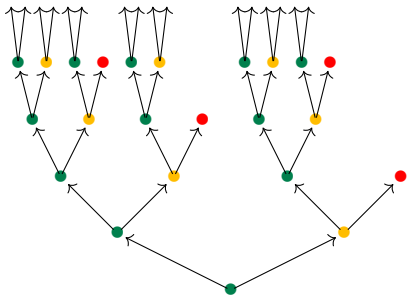
such morphism associates its codomain and then the set of morphisms emanating from that, and so on. The passback map simply takes a composable sequence of morphisms and composes them.

Exercise 5.157. Consider the walking arrow category $\mathcal{W} = \boxed{\bullet \to \bullet}$. Draw the cofunctor $\mathcal{W} \to \mathcal{T}_{y^2+y}$. ◊

Dynamical systems and graph fibrations. In **??**, there's a certain relationship we can see between the graph we associate to the dynamical system $Sy^S \to p$, namely



and the trees (which are also graphs) that its mate $Sy^S \to \mathcal{T}_p$ associates to each element of S, e.g. for the green dot:



Indeed, there is a map of graphs from the latter to the former, which sends all the green dots in the tree to the green dot in the dynamical system, etc. This map of graphs is a kind of *fibration*, or maybe we should say op-fibration, in the sense that the set of arrows emanating from every dot in the tree is in bijection with the set of arrows emanating from its image in the dynamical system graph.

Exercise 5.158.

1. Draw the other two trees associated to the dynamical system in **??**.
2. Do they also have an op-fibration down to the dynamical system graph?
3. Are these op-fibrations special in any way? That is, are they unique, or have any universal property? ◊

Replacing Sy^S by another comonoid. For any interface p, we defined a dependent dynamical system—also called a generalized Moore machine—to be a set S and a polynomial map $Sy^S \to p$. But now it seems that what really makes this work is that Sy^S underlies a comonoid. This suggests that we could instead have defined a

dependent dynamical system to be a comonoid $\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$ together with a map $\mathfrak{c} \to p$. What are the similarities and differences?

Here are some similarities. We still get a cofunctor $F\colon \mathcal{C} \to \mathscr{T}_p$, so we associate a p-tree to each object in $\mathcal{C}$ and pass back paths out of its root to morphisms in $\mathcal{C}$. In terms of dynamics, we would think of objects in $\mathcal{C}$ as internal states. We still have the situation from (**??**), meaning that for every state $c \in \mathcal{C}$, we get a position $i \coloneqq F_1(c)$ in $p(1)$, and for every direction $d \in p[i]$ there we get a new state $\mathrm{cod}(F_c^\sharp(d)) \in \mathcal{C}$.

But in fact we get a little more from F, and this is where the differences come in. Namely, given a direction $d \in p[i]$, we get the morphism $F_c^\sharp(d)$ itself. In the state category Sy^S, there is a unique morphism between every two objects, so this passed-back morphism carries no data beyond what its codomain is. But for a more general comonoid $\mathcal{C}$, the morphisms *do* carry data.

Thus we can think of a map $\mathfrak{c} \to p$ as a dynamical system that "records its history". That is, given a path $\mathscr{T}_p$, a sequence of inputs to our dynamical system, we get a morphism in $\mathcal{C}$. If, unlike in a state category Sy^S, there are multiple morphisms between objects, we will know which one was actually taken by the system.

This seems like a nice generalization of dynamical systems—history-recording dynamical systems—and may have some use. However, we will see in **??** that there are strpmg theoretical reason to emphasize the ahistorical state categories Sy^S. For one thing, the category of all such Sy^S-style dynamical systems on p forms a topos for any $p \in \mathbf{Poly}$.

5.3.5 Some math about cofree comonoids

Proposition 5.159. For every polynomial p, the cofree category $\mathscr{T}_p$ is free on a graph. That is, there is a graph G_p whose associated free category in the usual sense (the category of vertices and paths in G_p) is isomorphic to $\mathscr{T}_p$.

Proof. For vertices, we let V_p denote the set of p-trees,

$$V_p \coloneqq \mathsf{tree}_p(1).$$

For arrows we use the map $\pi\colon \mathsf{tree}_p \to p$ from **??** to define

$$A_p \coloneqq \sum_{t \in \mathsf{tree}_p(1)} p[\pi_1(t)]$$

In other words A_p is the set $\{d \in p[\pi_1(t)] \mid t \in \mathsf{tree}_p\}$ of directions in p that emanate from the root corolla of each p-tree. The source of (t, d) is t and the target is $\mathrm{cod}(\pi_t^\sharp(d))$. It is clear that every morphism in $\mathscr{T}_t$ is the composite of a finite sequence of such morphisms, completing the proof. □

Corollary 5.160. Let p be a polynomial and $\mathscr{T}_p$ the cofree comonoid. Every morphism in $\mathcal{C}_p$ is both monic and epic.

Proof. The free category on a graph always has this property, so the result follows from **??**. □

Proposition 5.161. The cofree functor $p \mapsto \mathscr{T}_p = (\mathsf{tree}_p, \mathrm{root}, \mathrm{focus})$ is lax monoidal; in particular there is a map of polynomials $y \to \mathsf{tree}_y$, and for any $p, q \in \mathbf{Poly}$ there is a natural map

$$\mathsf{tree}_p \otimes \mathsf{tree}_q \to \mathsf{tree}_{p \otimes q}.$$

satisfying the usual conditions.

Proof. By **??**, the left adjoint $U \colon \mathbf{Cat}^\sharp \to \mathbf{Poly}$ is strong monoidal. A consequence of Kelly's doctrinal adjunction theorem **[kelly1974doctrinal]** says that the right adjoint of an op-lax monoidal functor is lax monoidal. □

Exercise 5.162.

1. What polynomial is tree_y?
2. What is the map $y \to \mathsf{tree}_y$ from **??**?
3. Explain in words how to think about the map $\mathsf{tree}_p \otimes \mathsf{tree}_q \to \mathsf{tree}_{p \otimes q}$ from **??**, for arbitrary $p, q \in \mathbf{Poly}$. ◊

Proposition 5.163. The additive monoid $y^{\mathbb{N}}$ of natural numbers has a $\times$-monoid structure in $\mathbf{Cat}^\sharp$.

Proof. The right adjoint $p \mapsto \mathscr{T}_p$ preserves products, so $y^{\mathrm{List}(n)} \cong \mathscr{T}_{y^n}$ is the n-fold product of $y^{\mathbb{N}}$ in $\mathbf{Cat}^\sharp$. We thus want to find cofunctors $e \colon y \to y^{\mathbb{N}}$ and $m \colon y^{\mathrm{List}(2)} \to y^{\mathbb{N}}$ that satisfy the axioms of a monoid.

The unique polynomial map $y \to y^{\mathbb{N}}$ is a cofunctor (it is the mate of the identity $y \to y$). We take m to be the mate of the polynomial map $y^{\mathrm{List}(2)} \to y$ given by the list $[1, 2]$. One can check by hand that these definitions make $(y^{\mathbb{N}}, e, m)$ a monoid in $(\mathbf{Cat}^\sharp, y, \times)$. □

Proposition 5.164. The functor

$$\mathbf{Cat}^\sharp(-, y^{\mathbb{N}}) \colon \mathbf{Cat}^\sharp \to \mathbf{Mon}^{\mathrm{op}}$$

represented by $y^{\mathbb{N}}$ is right adjoint to the inclusion $\mathbf{Mon}^{\mathrm{op}} \to \mathbf{Cat}^\sharp$ from **??**.

Proof. We saw that $y^{\mathbb{N}}$ is a monoid object in **??**. A cofunctor $\mathcal{C} \nrightarrow y^{\mathbb{N}}$ is a policy in $\mathcal{C}$: it assigns an outgoing morphism to each object of $\mathcal{C}$. Any two such trajectories can be multiplied: we simply do one and then the other; this is the monoid operation. The policy assigning the identity to each object is the unit of the monoid. Let's denote this functor by $T \coloneqq \mathbf{Cat}^\sharp(-, y^{\mathbb{N}})$.

Let $\mathcal{C}$ be a category and $(M, e, *)$ a monoid. A cofunctor $F\colon \mathcal{C} \nrightarrow y^M$ has no data on objects; it is just a way to assign to each $c \in \mathcal{C}$ and $m \in M$ a morphism $F_c^\sharp(m)\colon c \to c'$ for some $c' \coloneqq \mathrm{cod}(F_c^\sharp(m))$. This assignment must send identities to identities and composites to composites: given $m' \in M$ we have $F_c^\sharp(m \fatsemi m') = F_c^\sharp(m) \fatsemi F_{c'}^\sharp(m')$. This is exactly the data of a monoid morphism $M \to T(\mathcal{C})$: it assigns to each $m \in M$ a policy in $\mathcal{C}$, preserving unit and multiplication. □

Proposition 5.165. There is a commutative square of left adjoints

$$\begin{array}{ccc} \mathbf{Mon}^{\mathrm{op}} & \xrightarrow{U} & \mathbf{Set}^{\mathrm{op}} \\ {\scriptstyle y^-}\downarrow & & \downarrow{\scriptstyle y^-} \\ \mathbf{Cat}^\sharp & \xrightarrow[U]{} & \mathbf{Poly} \end{array}$$

where the functors denoted U are forgetful functors.

Proof. Using the fully faithful functor $y^-\colon \mathbf{Mon}^{\mathrm{op}} \leftrightarrows \mathbf{Cat}^\sharp$ from **??**, it is easy to check that the above diagram commutes.

The free-forgetful adjunction $\mathbf{Set} \leftrightarrows \mathbf{Mon}$ gives an opposite adjunction $\mathbf{Set}^{\mathrm{op}} \leftrightarrows \mathbf{Mon}^{\mathrm{op}}$, where U is now left adjoint. We saw that $y^-\colon \mathbf{Set}^{\mathrm{op}} \to \mathbf{Poly}$ is a left adjoint in Proposition 4.178, that $U\colon \mathbf{Cat}^\sharp \to \mathbf{Poly}$ is a left adjoint in **??**, and that $y^-\colon \mathbf{Mon} \to \mathbf{Cat}^\sharp$ is a left adjoint in **??**. □

Chapter 6

Data dynamics

6.1 Introduction

In the previous chapter we saw that comonoids in **Poly** are categories, but that morphisms of comonoids are not functors; they're called cofunctors. If someone were to ask "which are better, functors or cofunctors", the answer is clear. Functors are fundamental to mathematics itself, relating branches from set theory to logic to algebra to measure theory, etc. Cofunctors don't have anywhere near that sort of reach in terms of applicability. Still they provide an interesting way to compare categories, as well as new invariants of categories, like the monoid of direction fields on a category.

In this chapter we'll consider another kind of morphism between comonoids in **Poly**, i.e. categories, and one that is a bit more familiar than cofunctors. Namely, we'll consider the bimodules between comonoids. One might want to call them bi-comodules, but this name is just a bit too long and the name bimodule is not ambiguous, so we'll go with it. So why do I say they're more familiar?

It turns out that bimodules between comonoids in **Poly** (categories) are also important objects of study in category theory. If $\mathcal{C}$ and $\mathcal{D}$ are categories, Richard Garner showed that a bimodule between them can be identified with what's known as a *parametric right adjoint* between the associated copresheaf categories $\mathcal{C}$-**Set** and $\mathcal{D}$-**Set**. Parametric right adjoints, or *pra*'s come up in ∞-category theory, but they also have a much more practical usage: they are the so-called *data migration functors*.

Indeed, we'll make due on a claim we made in **??**, that there is a strong connection between the basic theory of databases and the theory of bimodules in **Poly**. Databases have two parts: they have a schema, a specification of various types and relationships between them, and an instance, which is actual data sorted into those types and having those relationships. As we'll see, one can formalize the schema as a category $\mathcal{C}$ and the instance as a functor $I\colon \mathcal{C} \to \mathbf{Set}$. Here's an example of a theorem we'll prove:

Theorem 6.1. For a comonoid $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$, also understood as a category $\mathcal{C}$, the following categories are equivalent:

1. functors $\mathcal{C} \to$ **Set**;
2. discrete opfibrations over $\mathcal{C}$;
3. cartesian cofunctors to $\mathscr{C}$;
4. linear left $\mathscr{C}$-modules;
5. constant left $\mathscr{C}$-modules;
6. $(\mathscr{C}, 0)$-bimodules;
7. representable right $\mathscr{C}$-modules;
8. $\mathscr{C}$-coalgebras (sets with a coaction by $\mathscr{C}$).

Moreover, up to isomorphism, a $\mathscr{C}$-coalgebra can be identified with a dynamical system with comonoid interface $\mathscr{C}$.

The proof will be given in **??**.

The plan of the chapter is as follows. We'll begin in **??** by reviewing copresheaves on a category, and their relationship to databases and dynamical systems. Then in **??** we'll prove a number of theoretical results, including **??** and Garner's "bimodules are parametric right adjoints" result. We'll continue to give intuition and applications in database and dynamical systems theory. Finally in **??** we'll provide some looser discussion and lay out some open questions.

6.2 Copresheaves, databases, and dynamical systems

Let $\mathcal{C}$ be a small category. One of the most important constructions in category theory is that of the category of copresheaves on $\mathcal{C}$.[1] This is the category

$$\mathcal{C}\text{-}\mathbf{Set} \coloneqq \mathbf{Fun}(\mathcal{C}, \mathbf{Set})$$

whose objects are functors $\mathcal{C} \to$ **Set** and whose morphisms are natural transformations between them.

Example 6.2. Suppose $(G, e, *)$ is a monoid (e.g. a group). In a first course on abstract algebra, one encounters the notion of a *G-set*, which is a set X together with a G action: for every element $g \in G$ we get a function $\alpha_g \colon X \to X$; we might write $\alpha_g(x)$ as $g \cdot x$. To be a G-action, the $\cdot$ operation needs to satisfy two rules: $e \cdot x = x$ and $g \cdot (h \cdot x) = (g * h) \cdot x$. A morphism between two G-sets (sets X and Y, each equipped with a G-action) is just a function $f \colon X \to Y$ that satisfies a single rule: $f(g \cdot x) = g \cdot (f(x))$ for all $g \in G$ and $x \in X$.

Now recall that any monoid (e.g. a group) G can be understood as a category with one object, let's call our category $\mathcal{G}$ and the unique object $\blacktriangle$. The elements of G,

[1] Many would say that presheaves on $\mathcal{C}$ are more fundamental, but since the notions are equivalent—just use $\mathcal{C}^{\mathrm{op}}$ to switch between them—we will consider the difference moot. We will focus on copresheaves.

including the identity and the multiplication, are encoded as the morphisms ▲ → ▲ in $\mathcal{G}$.

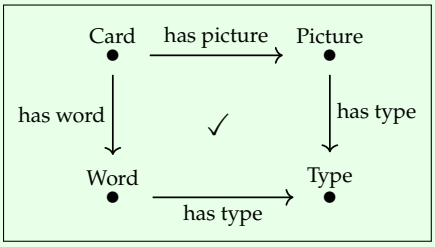
It turns out that G-sets are precisely functors $F\colon \mathcal{G} \to \mathbf{Set}$: the set $F(▲)$ is our X above, and since the elements of G are now morphisms $g\colon ▲ \to ▲$, the functor F sends them to functions $F(g)\colon X \to X$; this is the $g \cdot -$ operation. The axioms of a functor—preservation of identities and compositions—ensure the two rules of the $\cdot$ operation. Finally, morphisms between G-sets are exactly the natural transformations between functors; the naturality condition becomes the rule $f(g \cdot x) = g \cdot (f(x))$ we saw above.

G-sets are copresheaves on the associated one-object category $\mathcal{G}$.

Example 6.3 (Clue game). There is a game that is played in the US, called Clue, where each players is a detective trying to solve a murder mystery. The game consists of a board together with a deck of several cards, and on each card there is both some printed words and a picture. For one card, the words might say "Ms. White" and the picture shows a woman; for another card, the words might say "knife" and the picture shows a knife. Every card has a type: it is either a room card, a weapon card, or a person card; e.g. the Ms. White card is a person card and the knife card is a weapon card.

Now consider the commutative square category whose objects are labeled as follows:

Sq :=



A functor $D\colon \mathbf{Sq} \to \mathbf{Set}$ would be a possible Clue deck: a set of cards, each of which has an associated picture, some associated words, and the words and picture both have the same type (e.g. room, weapon, person, whatever). Of course, a priori, D is just a commutative square of sets and functions, but we'll refer to elements of $D(\text{Card})$ as cards, etc.

Here we will give our own version of the game and use it to illustrate databases. It will be a running example throughout the chapter. We start with a functor Deck$\colon \mathbf{Sq} \to$

Set, written as a database instance with schema **Sq**:

Deck Card			
Card	**has word**	**has picture**	**has type**
W	White	○	Color
B	Black	●	Color
C	Circle	O	Object
T	Table	Π	Object
P	Person	☥	Object
H	Here	·	Place
U	Up	↑	Place
D	Down	↓	Place
L	Left	←	Place
R	Right	→	Place

(6.4)

Deck Picture	
Picture	**has type**
○	Color
●	Color
O	Object
Π	Object
☥	Object
·	Place
↑	Place
↓	Place
←	Place
→	Place

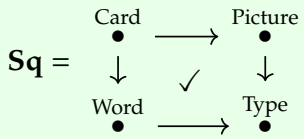
Deck Word	
Word	**has type**
White	Color
Black	Color
Circle	Object
Table	Object
Person	Object
Here	Place
Up	Place
Down	Place
Left	Place
Right	Place

Deck Type
Type
Color
Object
Place

Exercise 6.5.

1. With the instance Deck : **Sq** → **Set**, what set is Deck(Card)?
2. What set is Deck(Word)?
3. What is the function Deck(has word) : Deck(Card) → Deck(Card)?
4. Do you think there is anything mathematically-special about the database instance presented in **??**? That is, as a functor **Sq** → **Set**, does it seem relatively generic, or does something about it seem special?
5. If you'd say it's not special, why not? If you'd say it is special, give another instance (functor **Sq** → **Set**, written in database form) that is not special in this way. ◊

Example 6.6. Continuing with **??**, we again consider the commutative square category



Let $P := \{b, w\}^{100\times 100}$ be a set whose elements we call (black and white) pictures, let $W :=$ List({A,a, . . . , Z,z}) be a set whose elements we call words, and let $T :=$ {Color,Object,Place} be a set whose elements we call types. We can thus create a database instance $J\colon \mathbf{Sq} \to \mathbf{Set}$ sending

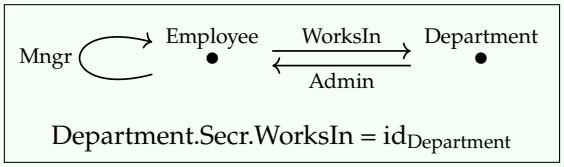
$$J(\text{Type}) := T, \quad J(\text{Word}) := W \times T, \quad J(\text{Picture}) := P \times T, \quad J(\text{Card}) := P \times W \times T$$

with the maps in **Sq** sent to the evident product projections. We will later see in **??** where these products and projections come from.

For now, we again consider our clue game. If we want to think of our pictures, e.g. 🧍, really as pictures or our words, e.g. "Person", really as words, then we need to be given a map of database instances—a natural transformation—of the form $\tau\colon \text{Deck} \to J$. In particular, τ assigns every card in the deck a word, a picture, and a type.

Exercise 6.7. Consider the category $\mathcal{C}$ shown here:

$\mathcal{C} :=$



It is generated by two objects, three morphisms, and the equation shown above.

1. What is its emanation polynomial?

Consider now the database instance shown here:

Employee	**WorksIn**	**Mngr**
Alice	IT	Alice
Bobby	IT	Alice
Carla	Sales	Carla

Department	**Admin**
IT	Bobby
Sales	Carla

Consider this database instance as a functor $S\colon \mathcal{C} \to \mathbf{Set}$.

2. Say what sets S assigns the two objects.
3. Say what functions S assigns the three generating morphisms. ◊

Definition 6.8 (Discrete opfibration)**.** Let $\mathcal{C}$ be a category. A pair $(\mathcal{S}, \pi)$, where $\mathcal{S}$ is a category and $\pi\colon \mathcal{S} \to \mathcal{C}$ is a functor, is called a *discrete opfibration over* $\mathcal{C}$ if it satisfies the

following condition.

- for every object $s \in \mathcal{S}$, object $c' \in \mathcal{C}$, and morphism $f : \pi(s) \to c'$ there exists a unique object $s' \in \mathcal{S}$ and morphism $\bar{f} : s \to s'$ such that $\pi(s') = c'$ and $\pi(\bar{f}) = f$.

$$\begin{array}{ccc} s & \overset{\bar{f}}{\dashrightarrow} & s' \\ {\scriptstyle\pi}\downarrow & & \downarrow{\scriptstyle\pi} \\ \pi(s) & \xrightarrow[f]{} & c' \end{array}$$

A *morphism* $(\mathcal{S}, \pi) \to (\mathcal{S}', \pi')$ between discrete opfibrations over $\mathcal{C}$ is a functor $F : \mathcal{S} \to \mathcal{S}'$ making the following triangle commute:

$$\begin{array}{ccc} \mathcal{S} & \xrightarrow{F} & \mathcal{S}' \\ & {\scriptstyle\pi}\searrow \quad \swarrow{\scriptstyle\pi'} & \\ & \mathcal{C} & \end{array} \tag{6.9}$$

We denote the category of discrete opfibrations into $\mathcal{C}$ by $\mathbf{dopf}(\mathcal{C})$.

Exercise 6.10. Show that if $F : \mathcal{S} \to \mathcal{S}'$ is a functor making the triangle (**??**) commute, then F is also a discrete opfibration. ◊

Exercise 6.11. Suppose $\pi : \mathcal{S} \to \mathcal{C}$ is a discrete opfibration and $i \in \mathcal{S}$ is an object. With notation as in **??**, show the following:

1. Show that the lift $\bar{\mathrm{id}}_{\pi(i)} = \mathrm{id}_i$ of the identity on $\pi(i)$ is the identity on i.
2. Show that for $f : \pi(i) \to c$ and $g : c \to c'$, we have $\bar{f} \fatsemi \bar{g} = \overline{f \fatsemi g}$.
3. Show that π is a cofunctor. ◊

Definition 6.12 (Category of elements $\int^{\mathcal{C}} I$)**.** Given a functor $I : \mathcal{C} \to \mathbf{Set}$, its category of elements $\int^{\mathcal{C}} I$ is defined to have objects

$$\mathrm{Ob}(\textstyle\int^{\mathcal{C}} I) \coloneqq \{(c, x) \mid c \in \mathrm{Ob}(\mathcal{C}), x \in I(c)\}$$

and given two objects (c, x) and (c', x'), the hom-set is given by

$$\mathrm{Hom}((c, x), (c', x')) \coloneqq \{f : c \to c' \mid I(f)(x) = x'\}.$$

Identities and composites in $(\int^{\mathcal{C}} I)$ are inherited from $\mathcal{C}$.

There is a functor $\pi : (\int^{\mathcal{C}} I) \to \mathcal{C}$ sending $\pi(c, x) \coloneqq c$ and $\pi(f) \coloneqq f$.

Exercise 6.13. Show that if $I\colon \mathcal{C} \to \mathbf{Set}$ is a functor then the functor $\pi\colon (\int^{\mathcal{C}} I) \to \mathcal{C}$ defined in **??** is a discrete opfibration, as in **??**. ◊

Exercise 6.14. Draw the category of elements for the functor $S\colon \mathcal{C} \to \mathbf{Set}$ shown in **??**. ◊

Exercise 6.15. Suppose that $I, J\colon \mathcal{C} \to \mathbf{Set}$ are functors and $\alpha\colon I \to J$ is a natural transformation.

1. Show that α induces a functor $(\int^{\mathcal{C}} I) \to (\int^{\mathcal{C}} J)$.
2. Show that it is a morphism of discrete opfibrations in the sense of **??**.
3. Show that this construction is functorial. We denote this functor by
$$\textstyle\int^{\mathcal{C}} : \mathcal{C}\text{-}\mathbf{Set} \to \mathbf{dopf}(\mathcal{C})$$

◊

Exercise 6.16. Let G be a graph, and let $\mathcal{G}$ be the free category on it. Show that for any functor $S\colon \mathcal{G} \to \mathbf{Set}$, the category $\int^{\mathcal{G}} S$ of elements is again free on a graph. ◊

Proposition 6.17. Let $\mathcal{C}$ be a category. The following are equivalent:

1. the category $\mathcal{C}$-**Set** of functors $\mathcal{C} \to \mathbf{Set}$,
2. the category of discrete opfibrations over $\mathcal{C}$,
3. the category of cartesian cofunctors into $\mathcal{C}$.

In fact the latter two are isomorphic.

Proof. By **??** we have a functor $\int^{\mathcal{C}} : \mathcal{C}\text{-}\mathbf{Set} \to \mathbf{dopf}(\mathcal{C})$. There is a functor going back: given a discrete opfibration $\pi\colon \mathcal{S} \to \mathcal{C}$, we define $(\partial\pi)\colon \mathcal{C} \to \mathbf{Set}$ on $c \in \mathrm{Ob}(\mathcal{C})$ and $f\colon c \to c'$ by

$$\begin{aligned}(\partial\pi)(c) &\coloneqq \{s \in \mathcal{S} \mid \pi(s) = c\}\\ (\partial\pi)(f)(s) &\coloneqq \bar{f}(s).\end{aligned}$$

On objects, the roundtrip $\mathcal{C}$-**Set** $\to \mathcal{C}$-**Set** sends $I\colon \mathcal{C} \to \mathbf{Set}$ to the functor

$$\begin{aligned} c \mapsto \{s \in \textstyle\int^{\mathcal{C}} I \mid \pi(s)\\ = \{(c, x) \mid x \in I(c)\} \qquad\qquad &= I(c).\end{aligned}$$

The roundtrip $\mathbf{dopf}(\mathcal{C}) \to \mathbf{dopf}(\mathcal{C})$ sends $\pi\colon \mathcal{S} \to \mathcal{C}$ to the discrete opfibration whose object set is $\{(c, s) \in \mathrm{Ob}(\mathcal{C}) \times \mathrm{Ob}(\mathcal{S}) \mid \pi(s) = c\}$ and this set is clearly in bijection with $\mathrm{Ob}(\mathcal{S})$. Proceeding similarly, one defines an isomorphism of categories $\mathcal{S} \cong \int^{\mathcal{C}} \partial\pi$.

The above correspondence is well-known; it remains to address the relationship between (2) and (3). A cartesian cofunctor $(\varphi_1, \varphi^\sharp)\colon \mathcal{S} \nrightarrow \mathcal{C}$ gives a function $\varphi\colon \mathrm{Ob}(\mathcal{S}) \to \mathrm{Ob}(\mathcal{C})$ and for each $s \in \mathcal{S}$ an isomorphism

$$\varphi^\sharp_s\colon \mathcal{C}[\varphi_1(s)] \xrightarrow{\cong} \mathcal{S}[s]$$

between the set of $\mathcal{S}$-morphisms emanating from s and the set of $\mathcal{C}$-morphisms emanating from $\varphi_1(s)$. This isomorphism respects identities, codomains, and composites. As such we can define a functor that acts as φ_1 on objects and $(\varphi^\sharp)^{-1}$ on morphisms, and it is easily checked to be a discrete opfibration.

Finally, given a discrete opfibration $\pi\colon \mathcal{S} \to \mathcal{C}$, we define $\varphi_1 \coloneqq \mathrm{Ob}(\pi)$ to be its on-objects part, and for any $s \in \mathrm{Ob}(\mathcal{S})$ and emanating morphism $f \in \mathcal{C}[\varphi_1(s)]$, we define $\varphi^\sharp_s(f) \coloneqq \bar{f}$ to be the lift guaranteed by **??**. The conversions between discrete opfibrations and cartesian cofunctors are easily seen to be functorial and the roundtrips are identities. □

Recall from **??** that a dynamical system on a category $\mathcal{C}$ consists of a set S and a cofunctor $Sy^S \nrightarrow \mathcal{C}$ from the state category on S to $\mathcal{C}$.

Proposition 6.18 (Database instances are dynamical systems)**.** Up to isomorphism, discrete opfibrations into $\mathcal{C}$ can be identified with dynamical systems on $\mathcal{C}$.

In case it isn't clear, this association is only functorial on the groupoid of objects and isomorphisms.

Proof. Given a discrete opfibration $\pi\colon \mathcal{S} \to \mathcal{C}$, take $S \coloneqq \mathrm{Ob}(\mathcal{S})$ and define $(\varphi_1, \varphi^\sharp)\colon Sy^S \to \mathfrak{c}$ by $\varphi_1 = \pi$ and with $\varphi^\sharp$ given by the lifting: $\varphi(g) \coloneqq \hat{g}$ as in **??**. One checks using **??** that this defines a cofunctor.

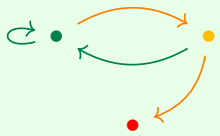
Conversely, given a cofunctor $(\varphi_1, \varphi^\sharp)\colon Sy^S \to \mathfrak{c}$, the function φ_1 induces a map of polynomials $Sy \to \mathfrak{c}$, and we can factor it as a vertical followed by a cartesian $Sy \to \mathfrak{s} \xrightarrow{\psi} \mathfrak{c}$. We can give $\mathfrak{s}$ the structure of a category such that ψ is a cofunctor; see **??**. □

Exercise 6.19. With notation as in **??**, complete the proof as follows.

1. Check that $(\varphi, \varphi^\sharp)$ defined in the first paragraph is indeed a cofunctor.
2. Find a comonoid structure on $\mathfrak{s}$ such that ψ is a cofunctor, as stated in the second paragraph.
3. Show that the two directions are inverse, up to isomorphism. ◊

Example 6.20. In **??** we had a dynamical system with $S \coloneqq \{\bullet, \bullet, \bullet\}$ and $p \coloneqq y^2 + y$, and

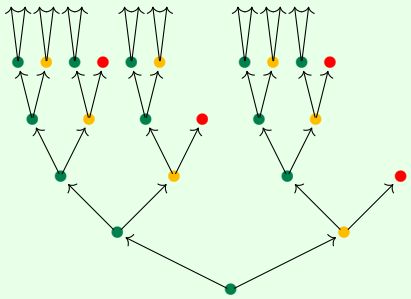
$f\colon Sy^S \to p$ from Exercise 4.129, depicted here again for your convenience:



(6.21)

The polynomial map f induces a cofunctor $F\colon Sy^S \nrightarrow \mathscr{T}_p$ from the state category on S to the category of p-trees. We can now see this as a database instance on $\mathscr{T}_p$, considered as a database schema.

The cofree category $\mathscr{T}_t$ is actually the free category on a graph, as we saw in **??**, and so the schema is easy. There is one table for each tree (object in $\mathscr{T}_p$), e.g. we have a table associated to this tree:



The table has two columns, say left and right, corresponding to the two arrows emanating from the root node. The left column refers back to the same table, and the right column refers to another table (the one corresponding to the yellow dot).

Again, there are infinitely many tables in this schema. Only three of them have data in them; the rest are empty. We know in advance that this instance has three rows in total, since $|S| = 3$.

Given a dynamical system $Sy^S \to p$, we extend it to a cofunctor $\varphi\colon Sy^S \nrightarrow \mathscr{T}_p$. By **????**, we can consider it as a discrete opfibration over $\mathscr{T}_p$. By **??** 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 (**??**).

Exercise 6.22. Give an example of a dynamical system on $p \coloneqq y^2 + y$ for which the corresponding database instance has a table with at least two rows. ◊

Exercise 6.23. A dynamical system with interface y is a map $Sy^S \to y$.

1. What is the corresponding database schema?
2. Explain what the corresponding database instance looks like.
3. In particular, how many total rows does it have?
4. Give an example with $|S| = 7$, displayed both as a dynamical system (with states and transitions) and as a database instance. ◊

6.3 Bimodules

One might think that the database story—or you could call it the copresheaves story—is somewhat tacked on to the main line here: cartesian cofunctors $\mathcal{S} \nrightarrow \mathcal{C}$ can be seen as database instances on $\mathcal{C}$, but perhaps this is just an incidental curiosity. In this section we'll see that in fact copresheaves, i.e. set-valued functors on categories, are a major aspect of the story.

In a monoidal category, one can not only consider monoids and comonoids, but also modules between these. For example, in the monoidal category of abelian groups under bilinear product, the monoid objects are rings and the bimodules are bimodules in the usual sense. Here we are interested in comonoids rather than monoids, so we should be interested in bi-comodules; we will just call these bimodules for linguistic convenience.

We will see that bimodules in **Poly** are equivalent to data-migration functors. Given comonoids/categories $\mathcal{C}$ and $\mathcal{D}$, a bimodule $\mathcal{C} \overset{m}{\triangleleft\!\!\longleftarrow\!\!\triangleleft} \mathcal{D}$ is a way of moving database instances on $\mathcal{D}$ to database instances on $\mathcal{C}$. One could call it a "$\mathcal{D}$-indexed union of conjunctive queries."

Definition 6.24 (Bimodule)**.** Let $\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$ and $\mathcal{D} = (\mathfrak{d}, \epsilon, \delta)$ be comonoids. A $(\mathcal{C}, \mathcal{D})$-*bimodule* (m, λ, ρ), denoted $\mathcal{C} \overset{m}{\triangleleft\!\!\longleftarrow\!\!\triangleleft} \mathcal{D}$ or in shorthand $\mathfrak{c} \overset{m}{\triangleleft\!\!\longleftarrow\!\!\triangleleft} \mathfrak{d}$,[a] consists of

1. a polynomial $m \in \mathbf{Poly}$,
2. a morphism $\mathfrak{c} \triangleleft m \overset{\lambda}{\leftarrow} m$, and
3. a morphism $m \overset{\rho}{\to} m \triangleleft \mathfrak{d}$,

satisfying the following commutative diagrams:

$$\begin{array}{ccc} \mathfrak{c} \triangleleft m & \xleftarrow{\lambda} & m \\ {\scriptstyle \epsilon \triangleleft m}\downarrow & \nearrow & \\ m & & \end{array} \qquad \begin{array}{ccc} \mathfrak{c} \triangleleft m & \xleftarrow{\lambda} & m \\ {\scriptstyle \delta \triangleleft m}\downarrow & & \downarrow{\scriptstyle \lambda} \\ \mathfrak{c} \triangleleft \mathfrak{c} \triangleleft m & \xleftarrow[\mathfrak{c} \triangleleft \lambda]{} & \mathfrak{c} \triangleleft m \end{array} \tag{6.25}$$

$$\begin{array}{ccc} m & \xrightarrow{\rho} & m \triangleleft \mathfrak{d} \\ & \searrow & \downarrow{\scriptstyle m \triangleleft \epsilon} \\ & & m \end{array} \qquad \begin{array}{ccc} m & \xrightarrow{\rho} & m \triangleleft \mathfrak{d} \\ {\scriptstyle \rho}\downarrow & & \downarrow{\scriptstyle m \triangleleft \delta} \\ m \triangleleft \mathfrak{d} & \xrightarrow[\rho \triangleleft \mathfrak{d}]{} & m \triangleleft \mathfrak{d} \triangleleft \mathfrak{d} \end{array} \tag{6.26}$$

$$\begin{array}{ccc} m & \xrightarrow{\rho} & m \triangleleft \mathfrak{d} \\ {\scriptstyle \lambda}\downarrow & & \downarrow{\scriptstyle \lambda \triangleleft \mathfrak{d}} \\ \mathfrak{c} \triangleleft m & \xrightarrow[\mathfrak{c} \triangleleft \rho]{} & \mathfrak{c} \triangleleft m \triangleleft \mathfrak{d} \end{array} \tag{6.27}$$

Just (m, λ) and the first line of diagrams is called a left $\mathcal{C}$-module, and similarly just (m, ρ) and the second line of diagrams is called a right $\mathfrak{D}$-module.

A *morphism* of $(\mathcal{C}, \mathcal{D})$-bimodules is a map $m \to n$ making the following diagram

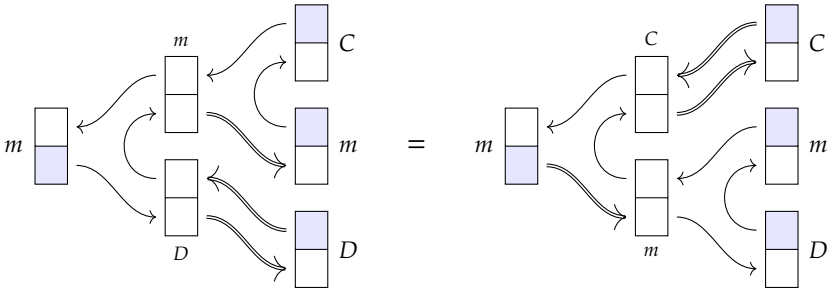
commute:

$$\begin{array}{ccccc} \mathfrak{c} \triangleleft m & \longleftarrow & m & \longrightarrow & m \triangleleft \mathfrak{d} \\ \downarrow & & \downarrow & & \downarrow \\ \mathfrak{c} \triangleleft n & \longleftarrow & n & \longrightarrow & n \triangleleft \mathfrak{d} \end{array}$$

We denote the category of $(\mathcal{C}, \mathcal{D})$-bimodules by ${}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}}$.

[a] Note that the symbol $\mathfrak{c} \triangleleft\!\!-\!\!\triangleleft \mathfrak{d}$ looks like it's going backwards, from $\mathfrak{d}$ to $\mathfrak{c}$; this is good because we'll see that this is the direction of data migration for a $(\mathcal{C}, \mathcal{D})$-bimodule. But the symbology is also mnemonically good because the $\triangleleft$'s go in the correct direction $\mathfrak{c} \triangleleft m \leftarrow m$ and $m \rightarrow m \triangleleft \mathfrak{d}$.

We draw the commutativity of **??** using polyboxes.

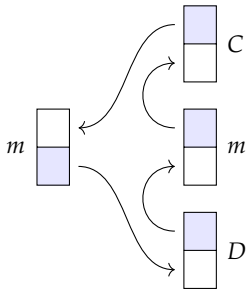


(6.28)

Exercise 6.29.

1. Draw the equations of **??** using polyboxes.
2. Draw the equations of **??** using polyboxes. ◊

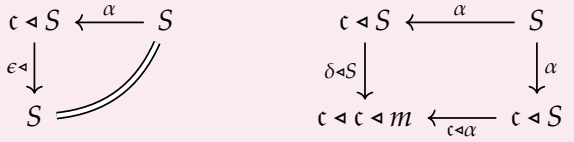
Note that **??** means that we can unambiguously write



Exercise 6.30. Let $\mathcal{C} = (\mathfrak{c}, \epsilon, \delta)$ be a category. Recall from **??** that y has a unique category structure.

1. Show that a left $\mathcal{C}$-module is the same thing as a $(\mathcal{C}, y)$-bimodule.
2. Show that a right $\mathcal{C}$-module is the same thing as a $(y, \mathcal{C})$-bimodule.
3. Show that every polynomial $p \in \mathbf{Poly}$ has a unique (y, y)-bimodule structure.
4. Show that there is an isomorphism of categories $\mathbf{Poly} \cong {}_{y}\mathbf{Mod}_{y}$. ◊

Definition 6.31 ($\mathscr{C}$-coalgebra)**.** Let $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ be a comonoid. A *$\mathscr{C}$-coalgebra* consists of a set S and a function $\alpha\colon S \to \mathfrak{c} \triangleleft S$, making the two diagrams below commute:



In other words, it is a left $\mathfrak{c}$-comodule whose carrier is constant on a set. A morphism is a function $S \to T$ commuting with α, just as for comodules; see **??**.

In order to as quickly as possible orient the reader to bimodules, we begin by proving the following.

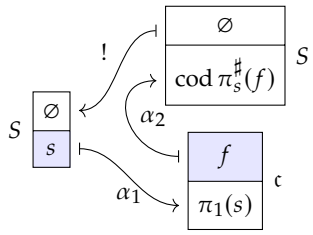
Theorem 6.32. For a comonoid $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ (category $\mathcal{C}$), the following categories are equivalent:

1. functors $\mathcal{C} \to$ **Set**;
2. discrete opfibrations over $\mathcal{C}$;
3. cartesian cofunctors to $\mathscr{C}$;
4. $\mathscr{C}$-coalgebras (sets with a coaction by $\mathscr{C}$);
5. constant left $\mathscr{C}$-modules;
6. $(\mathscr{C}, 0)$-bimodules;
7. linear left $\mathscr{C}$-modules; and
8. representable right $\mathscr{C}$-modules (opposite).

In fact, all but the first are isomorphic categories, and their core groupoid of is that of dynamical systems with interface $\mathcal{C}$.

Proof. $1 \simeq 2 \cong 3$**:** This was shown in **??**.

$3 \cong 4$**:** Given a cartesian cofunctor $(\pi_1, \pi^\sharp)\colon \mathcal{S} \nrightarrow \mathcal{C}$, let $S \coloneqq \mathrm{Ob}(\mathcal{S})$ and define a $\mathfrak{c}$-coalgebra structure $\alpha\colon S \to \mathfrak{c} \triangleleft S$ on an object $s \in \mathrm{Ob}(\mathcal{S})$ and an emanating morphism $f\colon \pi_1(s) \to c'$ in $\mathcal{C}$ by



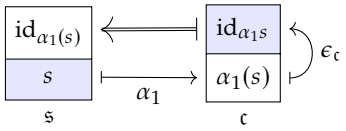
We check that this is indeed a coalgebra using properties of cofunctors. For identities in $\mathscr{C}$, we have

$$\begin{aligned} \alpha_2(s, \mathrm{id}_{\pi_1(s)}) &= \mathrm{cod}\, \pi^\sharp_s(\mathrm{id}_{\pi_1(s)}) \\ &= \mathrm{cod}\, \mathrm{id}_s = s \end{aligned}$$

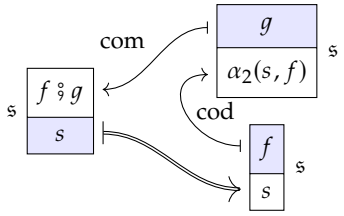
and for compositions in $\mathscr{C}$ we have

$$\begin{aligned}\alpha_2(s, f \fatsemi g) &= \operatorname{cod}\left(\pi^\sharp_s(f \fatsemi g)\right) \\ &= \operatorname{cod}\left(\pi^\sharp_s(f) \fatsemi \pi^\sharp_{\operatorname{cod}\pi^\sharp_s(f)} g\right) \\ &= \operatorname{cod}\left(\pi^\sharp_{\operatorname{cod}\pi^\sharp_s(f)} g\right) \\ &= \alpha_2(\alpha_2(s, f), g).\end{aligned}$$

Going backwards, if we're given a coalgebra $\alpha\colon S \to \mathfrak{c} \triangleleft S$, we obtain a function $\alpha_1\colon S \to \mathfrak{c} \triangleleft 1$ and we define $\mathfrak{s} := \alpha_1^* \mathfrak{c}$ and the cartesian map $\varphi := (\alpha_1, \mathrm{id})\colon \mathfrak{s} \to \mathfrak{c}$ to be the base change from Proposition 4.222. We need to show $\mathfrak{s}$ has a comonoid structure $(\mathfrak{s}, \epsilon, \delta)$ and that φ is a cofunctor. We simply define the counit $\epsilon\colon \mathfrak{s} \to y$ using α_1 and the counit on $\mathfrak{c}$:



We comultiplication $\delta\colon \mathfrak{s} \to \mathfrak{s} \triangleleft \mathfrak{s}$ using α_2 for the codomain, and using the composite $\fatsemi$ from $\mathfrak{c}$:



In **??** we check that $(\mathfrak{s}, \epsilon, \delta)$ really is a comonoid, that $(\alpha_1, \mathrm{id})\colon \mathfrak{s} \nrightarrow \mathfrak{c}$ is a cofunctor, that the roundtrips between cartesian cofunctors and coalgebras are identities, and that these assignments are functorial.

4 ≅ 5**:** This is straightforward and was mentioned in **??**.

5 ≅ 6**:** A right 0-module is in particular a polynomial $m \in$ **Poly** and a map $\rho\colon m \to m \triangleleft 0$ such that $(m \triangleleft \epsilon) \circ \rho = \mathrm{id}_m$. This implies ρ is monic, which itself implies by Proposition 4.180 that m must be constant since $m \triangleleft 0$ is constant. This makes $m \triangleleft \epsilon$ the identity, at which point ρ must also be the identity. Conversely, for any set M, the corresponding constant polynomial is easily seen to make the diagrams in (**??**) commute.

5 ≅ 7**:** By the adjunction in Proposition 4.16 and the fully faithful inclusion **Set** → **Poly** of sets as constant polynmials, Proposition 4.168, we have isomorphisms

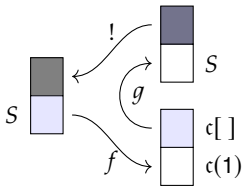
$$\mathbf{Poly}(Sy, \mathfrak{c} \triangleleft Sy) \cong \mathbf{Set}(S, \mathfrak{c} \triangleleft Sy \triangleleft 1) = \mathbf{Set}(S, \mathfrak{c} \triangleleft S) \cong \mathbf{Poly}(S, \mathfrak{c} \triangleleft S).$$

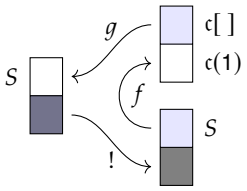
One checks easily that if $Sy \to \mathfrak{c} \triangleleft Sy)$ corresponds to $S \to \mathfrak{c} \triangleleft S$ under the above isomorphism, then one is a left module iff the other is.

$7 \cong 8$**:** By **??** we have a natural isomorphism

$$\mathbf{Poly}(Sy, \mathfrak{c} \triangleleft Sy) \cong \mathbf{Poly}(y^S, y^S \triangleleft \mathfrak{c}).$$

In pictures,





The last claim was proven in **??**. □

Exercise 6.33. Complete the proof of **??** ($3 \cong 4$) by proving the following.

1. Show that $(\mathfrak{s}, \epsilon, \delta)$ really is a comonoid.
2. Show that $(\alpha_1, \mathrm{id})\colon \mathfrak{s} \nrightarrow \mathfrak{c}$ is a cofunctor.
3. Show that the roundtrips between cartesian cofunctors and coalgebras are identities.
4. Show that the assignment of a $\mathscr{C}$-coalgebra to a cartesian cofunctor over $\mathcal{C}$ is functorial.
5. Show that the assignment of a cartesian cofunctor over $\mathcal{C}$ to a $\mathscr{C}$-coalgebra is functorial. ◊

Let $\mathcal{C}$ be a category. Under the above correspondence, the terminal functor $\mathcal{C} \to \mathbf{Set}$ corresponds to the identity discrete opfibration $\mathcal{C} \to \mathcal{C}$, the identity cofunctor $\mathscr{C} \to \mathscr{C}$, a certain left $\mathscr{C}$ module with carrier $\mathscr{C}(1)y$ which we call the *canonical left $\mathscr{C}$-module*, a certain constant left $\mathscr{C}$ module with carrier $\mathscr{C}(1)$ which we call the *canonical* $(\mathscr{C}, 0)$*-bimodule*, and a certain representable right $\mathscr{C}$-module with carrier $y^{\mathscr{C}(1)}$ which we call the *canonical right $\mathcal{C}$-module*.

Exercise 6.34. For any object $c \in \mathcal{C}$, consider the representable functor $\mathcal{C}(c, -)\colon \mathcal{C} \to \mathbf{Set}$. What does it correspond to as a

1. discrete opfibration over $\mathcal{C}$?
2. cartesian cofunctor to $\mathscr{C}$?
3. linear left $\mathscr{C}$-module?
4. constant left $\mathscr{C}$-module?
5. $(\mathscr{C}, 0)$-bimodule?
6. representable right $\mathscr{C}$-module?
7. dynamical system with comonoid interface $\mathscr{C}$? ◊

Exercise 6.35. We saw in **??** that the category ${}_{\mathcal{C}}\mathbf{Mod}_0$ of $(\mathcal{C}, 0)$-bimodule has a very nice structure: it's the topos of copresheaves on $\mathcal{C}$.

1. What is a $(0, \mathcal{C})$-bimodule?
2. What is ${}_0\mathbf{Mod}_{\mathcal{C}}$? ◊

Example 6.36. We continue with the Clue game from **??**. The database instance is encoded as:

$$\begin{pmatrix} 10 & & 10 \\ & + & \\ 10 & & 3 \end{pmatrix} \longrightarrow \begin{pmatrix} \begin{pmatrix} 10 & & 10 \\ & + & \\ 10 & & 3 \end{pmatrix}^4 & + & \begin{pmatrix} 10 & & 10 \\ & + & \\ 10 & & 3 \end{pmatrix}^2 \\ \\ \begin{pmatrix} 10 & & 10 \\ & + & \\ 10 & & 3 \end{pmatrix}^2 & + & \begin{pmatrix} 10 & & 10 \\ & + & \\ 10 & & 3 \end{pmatrix}^1 \end{pmatrix}$$

Or if you prefer, $33 \to 33^4 + 33^2 + 33^2 + 33$.

Recall from Proposition 4.222 that for any function $f : A \to B$, we have a base-change functor $f^* : B\mathbf{Poly} \to A\mathbf{Poly}$ and a cartesian morphism $f^*p \to p$ for any polynomial p and isomorphism $p(1) \cong B$.

Proposition 6.37. Let $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ be a comonoid and suppose that $\rho : m \to m \triangleleft \mathfrak{c}$ is a right $\mathscr{C}$-module. Then for any set A and function $f : A \to m \triangleleft 1$, the polynomial f^*m has an induced right module structure ρ_f fitting into the commutative square below:

$$\begin{array}{ccc} f^*m & \xrightarrow{\rho_f} & f^*m \triangleleft \mathfrak{c} \\ \downarrow & & \downarrow \\ m & \xrightarrow[\rho]{} & m \triangleleft \mathfrak{c} \end{array}$$

Proof. The pullback diagram to the left defines $f^*(m)$ and that to the right is its composition with $\mathfrak{c}$

$$\begin{array}{ccc} f^*m & \longrightarrow & m \\ \downarrow & \lrcorner & \downarrow \\ A & \xrightarrow[f]{} & m \triangleleft 1 \end{array} \qquad \begin{array}{ccc} f^*m \triangleleft \mathfrak{c} & \longrightarrow & m \triangleleft \mathfrak{c} \\ \downarrow & \lrcorner & \downarrow \\ A & \xrightarrow[f]{} & m \triangleleft 1 \end{array}$$

which is again a pullback by **????**. Now the map $\rho : m \to m \triangleleft \mathfrak{c}$ induces a map $\rho_f : f^*(m) \to f^*(m) \triangleleft c$; we claim it is a right module. It suffices to check that ρ_f interacts properly with ϵ and δ, which we leave to **??**. □

Exercise 6.38. Let $\mathfrak{c}, \epsilon, \delta)$, $\rho\colon m \to m \triangleleft \mathfrak{c}$, and $f\colon A \to m \triangleleft 1$ be as in **??**. Complete the proof of that proposition as follows:

1. Show that $\rho_f \fatsemi \epsilon = \mathrm{id}_m$
2. Show that $\rho_f \fatsemi (f^* m \triangleleft \delta) = \rho_f \fatsemi (\rho_f \triangleleft \mathfrak{c})$. ◊

Proposition 6.39. Let $\mathscr{C} = (\mathfrak{c}, \epsilon, \delta)$ be a comonoid in **Poly**. For any set G, the polynomial $y^G \triangleleft \mathfrak{c}$ has a natural right $\mathscr{C}$-module structure.

Proof. We use the map $(y^G \triangleleft \delta)\colon (y^G \triangleleft \mathfrak{c}) \to (y^G \triangleleft \mathfrak{c} \triangleleft \mathfrak{c})$. It satisfies the unitality and associativity laws because $\mathfrak{c}$ does. □

We can think of elements of G as "generators". Then if $i'\colon G \to \mathfrak{c} \triangleleft 1$ assigns to every generator an object of a category $\mathcal{C}$, then we should be able to find the free $\mathcal{C}$-set that i' generates.

Proposition 6.40. Functions $i'\colon G \to \mathfrak{c} \triangleleft 1$ are in bijection with positions $i \in y^G \triangleleft \mathfrak{c} \triangleleft 1$. Let $m \coloneqq i^*(y^G \triangleleft \mathfrak{c})$ and let ρ_i be the induced right $\mathscr{C}$-module structure from **??**. Then ρ_i corresponds to the free $\mathcal{C}$-set generated by i'.

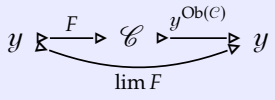
Proof. The polynomial m has the following form:

$$m \cong y^{\sum_{g \in G} \mathfrak{c}[i'(g)]}$$

In particular ρ_i is a representable right $\mathscr{C}$-module, and we can identify it with a $\mathcal{C}$-set by **??**. The elements of this $\mathcal{C}$-set are pairs (g, f), where $g \in G$ is a generator and $f\colon i'(g) \to \mathrm{cod}(f)$ is a morphism in $\mathcal{C}$ emanating from $i'(g)$. It is easy to see that the module structure induced by **??** is indeed the free one. □

Proposition 6.41 (Niu)**.** Let $\mathscr{C}$ be a comonoid. The category of left $\mathscr{C}$ modules is equivalent to the category of functors $\mathcal{C} \to$ **Poly**.

Proposition 6.42. For any functor $F\colon \mathcal{C} \to$ **Poly**, the limit polynomial $\lim_{c \in \mathcal{C}} F(c)$ is obtained by composing with the canonical right bimodule $y^{\mathrm{Ob}(\mathcal{C})}$



Proposition 6.43. Let $\mathscr{C}$ be a comonoid. For any set I and right $\mathscr{C}$-modules $(m_i)_{i \in I}$, the coproduct $m \coloneqq \sum_{i \in I} m_i$ has a natural right-module structure. Moreover, each representable summand in the carrier m of a right $\mathscr{C}$-module is itself a right-$\mathscr{C}$ module and m is their sum.

Proof. ** □

Proposition 6.44. If $m \in$ **Poly** is equipped with both a right $\mathscr{C}$-module and a right $\mathscr{D}$-module structure, we can naturally equip m with a $(\mathscr{C} \times \mathscr{D})$-module structure.

Proof. It suffices by **??** to assume that $m = y^M$ is representable. But a right $\mathscr{C}$-module with carrier y^M can be identified with a cofunctor $My^M \to \mathscr{C}$.

Thus if y^M is both a right-$\mathscr{C}$ module and a right-$\mathscr{D}$ module, then we have comonoid morphisms $\mathscr{C} \leftarrow My^M \to \mathscr{D}$. This induces a unique comonoid morphism $My^M \to (\mathscr{C} \times \mathscr{D})$ to the product, and we identify it with a right-$(\mathscr{C} \times \mathscr{D})$ module on y^M. □

6.3.1 Bimodules as data migration functors

When a polynomial

$$m \coloneqq \sum_{i \in m(1)} y^{m[i]}$$

is given the structure of a $(\mathcal{D}, \mathcal{C})$-bimodule, the symbols in that formula are given a hidden special meaning:

$$m(1) \in \mathcal{D}\text{-}\mathbf{Set} \qquad \text{and} \qquad m[i] \in \mathcal{C}\text{-}\mathbf{Set}$$

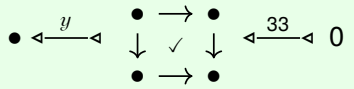
Thus $m(1)$ is a database instance on $\mathcal{D}$; in particular, each position in $i \in m(1)$ is a row in that instance. And each $m[i]$ is a database instance on $\mathcal{C}$; in particular, each direction $d \in m[i]$ is a row in that instance.

Before we knew about bimodule structures, what we called positions and directions—and what we often think of as outputs and inputs of a system—were understood as each forming an ordinary set. In the presence of a bimodule structure, the positions $m(1)$ have been organized into a $\mathcal{D}$-set and the directions $m[i]$ have been organized into a $\mathcal{C}$-set for each position i. We are listening for $\mathcal{C}$-sets and positioning ourselves in a $\mathcal{D}$-set.

Example 6.45. We continue with the Clue game from **????**. The database instance is encoded as a map

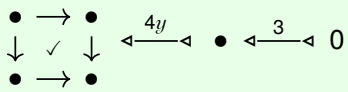
$$33 \to (y^4 + 2y^2 + y) \triangleleft 33.$$

In this example we will first grab the three-element set of types, and then we'll make a new 12-element instance of the commutative square $y^4 + 2y^2 + y$ that copies that three-element set four times.



The composite is the three-element set of types $\bullet \xleftarrow{3} 0$, where here 3 is shorthand

for {Color,Object,Place}. Now we can compose it with the "copy-all" map:



The result is the following **Sq**-set:

Copy Types Deck Card			
Card	**has word**	**has picture**	**has type**
Color	Color	Color	Color
Object	Object	Object	Object
Place	Place	Place	Place

Copy Types Deck Picture	
Picture	**has type**
Color	Color
Object	Object
Place	Place

Copy Types Deck Picture	
Picture	**has type**
Color	Color
Object	Object
Place	Place

Copy Types Deck Type
Type
Color
Object
Place

Now why would we want to do this? The answer is that the maps of bimodules from this instance to the Deck are exactly what are called the *suggestions* in the original Clue game. There are 30 of them:

$$\left\{\begin{array}{lllll}
(\circ, O, \cdot), & (\circ, O, \uparrow), & (\circ, O, \downarrow), & (\circ, O, \leftarrow), & (\circ, O, \rightarrow), \\
(\circ, \Pi, \cdot), & (\circ, \Pi, \uparrow), & (\circ, \Pi, \downarrow), & (\circ, \Pi, \leftarrow), & (\circ, \Pi, \rightarrow), \\
(\circ, \text{웃}, \cdot), & (\circ, \text{웃}, \uparrow), & (\circ, \text{웃}, \downarrow), & (\circ, \text{웃}, \leftarrow), & (\circ, \text{웃}, \rightarrow), \\
(\bullet, O, \cdot), & (\bullet, O, \uparrow), & (\bullet, O, \downarrow), & (\bullet, O, \leftarrow), & (\bullet, O, \rightarrow), \\
(\bullet, \Pi, \cdot), & (\bullet, \Pi, \uparrow), & (\bullet, \Pi, \downarrow), & (\bullet, \Pi, \leftarrow), & (\bullet, \Pi, \rightarrow), \\
(\bullet, \text{웃}, \cdot), & (\bullet, \text{웃}, \uparrow), & (\bullet, \text{웃}, \downarrow), & (\bullet, \text{웃}, \leftarrow), & (\bullet, \text{웃}, \rightarrow)
\end{array}\right\}$$

Exercise 6.46. Show that the 33-card deck from **??** factors—up to isomorphism—through a bimodule of the form $y^4 + 2y^2 + y \triangleleft\!\!-\!\!\triangleleft\ y^2 + y$, where $y^2 + y$ carries the walking arrow. ◊

6.3.2 Composing bimodules

Proposition 6.47 (Niu)**.** The composite of a linear left $\mathcal{C}$-module and a representable right $\mathcal{C}$-module is the set of natural transformations between the corresponding co-presheaves.

Example 6.48. Using **??** we can obtain the 30 clue suggestions from **??** as a composite of bimodules. (finish)

6.4 The proarrow equipment

6.5 Discussion and open questions

In this section, we lay out some questions that whose answers may or may not be known, but which were not known to us at the time of writing. They vary from concrete to open-ended, they are not organized in any particular way, and are in no sense complete. Still we hope they may be useful to some readers.

1. What can you say about the internal logic for the topos $\mathscr{T}_p\mathbf{Set}$ of dynamical systems with interface p, in terms of p?
2. How does the logic of the topos $\mathscr{T}_p$ help us talk about issues that might be useful in studying dynamical systems?
3. Morphisms $p \to q$ in **Poly** give rise to left adjoints $\mathscr{T}_p \to \mathscr{T}_q$ that preserve connected limits. These are not geometric morphisms in general; in some sense they are worse and in some sense they are better. They are worse in that they do not preserve the terminal object, but they are better in that they preserve every connected limit not just finite ones. How do these left adjoints translate statements from the internal language of p to that of q?
4. Consider the $\times$-monoids and $\otimes$-monoids in three categories: **Poly**, $\mathbf{Cat}^\sharp$, and **Mod**. Find examples of these comonoids, and perhaps characterize them or create a theory of them.
5. Is there a functor **Poly** has pullbacks, so one can consider the bicategory of spans in **Poly**. Is there a functor from that to **Mod** that sends $p \mapsto \mathscr{T}_p$?
6. Databases are static things, whereas dynamical systems are dynamic; yet we see them both in terms of **Poly**. How do they interact? Can a dynamical system read from or write to a database in any sense?
7. Can we do database aggregation in a nice dynamic way?
8. In the theory of polynomial functors, sums of representable functors **Set** $\to$ **Set**, what happens if we replace sets with homotopy types: how much goes through? Is anything improved?
9. Are there any functors **Set** $\to$ **Set** that aren't polynomial, but which admit a comonoid structure with respect to composition $(y, \triangleleft)$?
10. Characterize the monads in poly? They're generalizations of one-object operads (which are the Cartesian ones), but how can we think about them?
11. Both functors and cofunctors give left adjoint bimodules: for functors $F : \mathcal{D} \to \mathcal{C}$ we use the pullback Δ_F and for cofunctors $G : \mathcal{C} \nrightarrow \mathcal{D}$ we use the companion as in **??**. Can we characterize left adjoint bimodules in general?
12. What limits exist in $\mathbf{Cat}^\sharp$? Describe them combinatorially.
13. Since the forgetful functor $U : \mathbf{Cat}^\sharp \to \mathbf{Poly}$ is faithful, it reflects monomorphisms: if $f : \mathcal{C} \nrightarrow \mathcal{D}$ is a cofunctor whose underlying map on emanation polynomials is monic, then it is monic. Are all monomorphisms in $\mathbf{Cat}^\sharp$ of this form?
14. At first blush it appears that **Poly** may be suitable as the semantics of a language

for protocols. Develop such a language or showcase the limitations that make it impossible or inconvenient.

Chapter 7

Appendix H