Poly: an abundant categorical setting for mode-dependent dynamics

David I. Spivak

ACT2020
2020 July 7

Outline

A little personal history

Since a young age, I thought that math could help me think about reality.

- This reality; my own life; what's really going on right now.
- I wanted to create math that would help me think and ask questions.

A little personal history

Since a young age, I thought that math could help me think about reality.

- This reality; my own life; what's really going on right now.
- I wanted to create math that would help me think and ask questions.

Is everything *information*, made meaningful by structured relationships?

- At some point I thought that everything is information.
- I studied how information is stored and communicated, e.g. databases.

A little personal history

Since a young age, I thought that math could help me think about reality.

- This reality; my own life; what's really going on right now.
- I wanted to create math that would help me think and ask questions.

Is everything *information*, made meaningful by structured relationships?

- At some point I thought that everything is information.
- I studied how information is stored and communicated, e.g. databases.

Is everything *process*, interactive transforming of each others' products?

- I noticed that process didn't seem to fit well into databases.
- I studied how complex processes are formed from simpler ones.

A little personal history

Since a young age, I thought that math could help me think about reality.

- This reality; my own life; what's really going on right now.
- I wanted to create math that would help me think and ask questions.

Is everything *information*, made meaningful by structured relationships?

- At some point I thought that everything is information.
- I studied how information is stored and communicated, e.g. databases.

Is everything *process*, interactive transforming of each others' products?

- I noticed that process didn't seem to fit well into databases.
- I studied how complex processes are formed from simpler ones.

Shocking plot twist:

- These two worlds converge in **Poly**.
- I only have time to talk about dynamics today.

Plan for today

Today's plan:

- Recall some basics of **Poly**;
- Discuss how **Poly** models dynamical systems;
- Conclude with a brief summary.

Outline

Poly for experts

What I'll call the category **Poly** has many names.

- The free completely distributive category on one object;
- The full subcategory of $[\mathbf{Set}, \mathbf{Set}]$ spanned by functors that preserve connected limits;
- The full subcategory of $[\mathbf{Set}, \mathbf{Set}]$ spanned by coproducts of repr'bles;

Poly for experts

What I'll call the category **Poly** has many names.

- The free completely distributive category on one object;
- The full subcategory of $[\mathbf{Set}, \mathbf{Set}]$ spanned by functors that preserve connected limits;
- The full subcategory of $[\mathbf{Set}, \mathbf{Set}]$ spanned by coproducts of repr'bles;
- The "generalized lens category" associated to the canonical self-indexing $\mathbf{Set}/-\colon \mathbf{Set}^{\mathsf{op}} \to \mathbf{Cat}$ of $\mathbf{Set}$;
- The category of containers (in the sense of Michael Abbott).

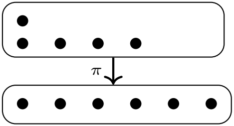
But let's make this easier.

What is a polynomial?

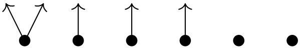
Algebraic

$$y^2 + 3y + 2$$

Bundle



Corolla forest



The category of polynomials

Easiest description: **Poly** $=$ "sums of representables functors **Set** $\to$ **Set**".

- For any set S, let $y^S := \mathbf{Set}(S, -)$, the functor *represented* by S.
- Def: a polynomial is a sum $p = \sum_{i \in I} y^{p_i}$ of representable functors.
- Def: a morphism of polynomials is a natural transformation.
- In **Poly**, $+$ is coproduct and $\times$ is product.

Notation

We said that a polynomial is a sum of representable functors

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

But note that $I \cong \sum_{i \in I} 1 = \sum_{i \in I} 1^{p_i} = p(1)$. So we can write

$$p \cong \sum_{i \in p(1)} y^{p_i}.$$

Bimorphic lenses are monomials

A *bimorphic lens* (Hedges) between set-pairs (S_1, T_1) and (S_2, T_2) is:

$$\begin{aligned} S_1 &\xrightarrow{get} S_2 \\ S_1 \times T_2 &\xrightarrow{put} T_1 \end{aligned} \tag{1}$$

Let **Lens** denote the cat'y with set-pairs as objects and lenses as morphisms.

There is an equivalence of categories $\mathbf{Lens} \cong \mathbf{Poly}_{monomials}$.

- Send $(S, T) \mapsto Sy^T$.
- Note, $\mathbf{Poly}(S_1 y^{T_1}, S_2 y^{T_2}) \cong \prod_{s \in S_1} S_2 \times T_1^{T_2}$, elements are as in (1).

So we can think of **Poly** as a generalized lens category.

Four interacting monoidal structures

We've seen two monoidal structures on **Poly** $(+, \times)$; there are two more.

- Dirichlet product $\otimes$; unit is y.
  - Let $p = \sum_{i \in p(1)} y^{p_i}$ and $q = \sum_{j \in q(1)} y^{q_j}$; compare:

$$p \times q \cong \sum_{i \in p(1)} \sum_{j \in q(1)} y^{p_i + q_j} \qquad \text{and} \qquad p \otimes q \cong \sum_{i \in p(1)} \sum_{j \in q(1)} y^{p_i q_j}.$$

Four interacting monoidal structures

We've seen two monoidal structures on **Poly** $(+, \times)$; there are two more.

- Dirichlet product $\otimes$; unit is y.
  - Let $p = \sum_{i \in p(1)} y^{p_i}$ and $q = \sum_{j \in q(1)} y^{q_j}$; compare:

$$p \times q \cong \sum_{i \in p(1)} \sum_{j \in q(1)} y^{p_i + q_j} \qquad \text{and} \qquad p \otimes q \cong \sum_{i \in p(1)} \sum_{j \in q(1)} y^{p_i q_j}.$$

- Composition product $\circ$; unit is y.
  - Just compose the functors; it's usual polynomial composition.
  - This monoidal structure is non-symmetric, $p \circ q \ncong q \circ p$.
  - It is a very interesting monoidal structure, as we'll see.

Composition monoidal structure $(\mathbf{Poly}, \circ, y)$

Let $p, q, \in$ **Poly** be polynomials. The formula for $p \circ q$ is

$$p \circ q \cong \sum_{i \in p(1)} \prod_{d \in p_i} \sum_{j \in q(1)} \prod_{e \in q_j} y.$$

Later we'll think about $p^{\circ n}$:

$$p^{\circ n} \cong \sum_{i_1 \in p(1)} \prod_{d_1 \in p_{i_1}} \sum_{i_2 \in p(1)} \prod_{d_2 \in p_{i_2}} \cdots \sum_{i_n \in p(1)} \prod_{d_n \in p_{i_n}} y$$

It's a length-n strategy: a choice of i_1, and for every d_1, a choice of i_2, etc.

Outline

Moore machines

Definition

Given sets A, B, an (A, B)-*Moore machine* consists of:

- a set S, elements of which are called *states*,
- a function $r\colon S \to B$, called *readout*, and
- a function $u\colon S \times A \to S$, called *update*.

It is *initialized* if it is equipped also with

- an element $s_0 \in S$, called the *initial state*.

We refer to A as the *input set*, B as the *output set*, and (A, B) as the *interface* of the Moore machine.

Moore machines

Definition

Given sets A, B, an (A, B)-*Moore machine* consists of:

- a set S, elements of which are called *states*,
- a function $r\colon S \to B$, called *readout*, and
- a function $u\colon S \times A \to S$, called *update*.

It is *initialized* if it is equipped also with

- an element $s_0 \in S$, called the *initial state*.

We refer to A as the *input set*, B as the *output set*, and (A, B) as the *interface* of the Moore machine.

Dynamics: an (A, B)-Moore machine (S, r, u, s_0) is a "stream transducer":

- Given a list/stream $[a_0, a_1, \ldots]$ of A's...
- let $s_{n+1} := u(s_n, a_n)$ and $b_n := r(s_n)$.
- We thus have obtained a list/stream $[b_0, b_1, \ldots]$ of B's.

Moore machines as lenses

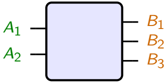
We can see Moore machines in terms of lenses / polynomials.

- An uninitialized Moore machine $r \colon S \to B$ and $u \colon S \times A \to S$ is:
  - A lens $(S, S) \to (B, A)$, i.e....
  - A map of polynomials $Sy^S \to By^A$.
- An initialized Moore machine also has a map $1 \to S$, so it is:
  - A composable pair of lenses $(1, 1) \to (S, S) \to (B, A)$, i.e....
  - a composable pair of polynomial maps $y \to Sy^S \to By^A$.

Depicting Moore machine interfaces

Here's how we depict interfaces (A, B) for Moore machines:

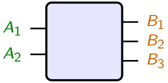
If, e.g. $A = A_1 \times A_2$ and $B = B_1 \times B_2 \times B_3$, we will instead draw:



Depicting Moore machine interfaces

Here's how we depict interfaces (A, B) for Moore machines:

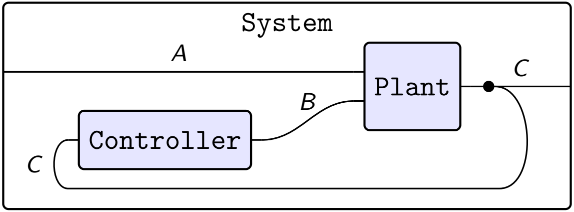
If, e.g. $A = A_1 \times A_2$ and $B = B_1 \times B_2 \times B_3$, we will instead draw:



In **Poly** these two interfaces are denoted By^A and $B_1 B_2 B_3 y^{A_1 A_2}$.

Wiring diagrams

Here's a picture of a wiring diagram:

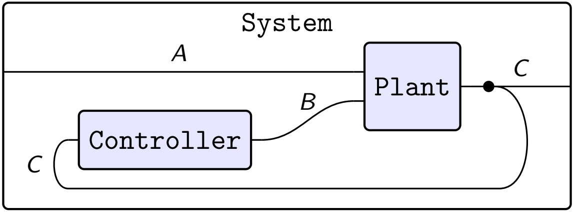


It includes three interfaces: `Controller`, `Plant`, and `System`.

$$\texttt{Controller} = By^{C} \qquad \texttt{Plant} = Cy^{AB} \qquad \texttt{System} = Cy^{A}$$

Wiring diagrams

Here's a picture of a wiring diagram:



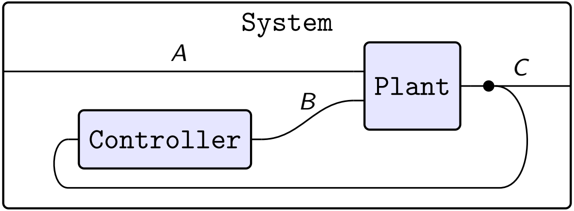
It includes three interfaces: `Controller`, `Plant`, and `System`.

$$\texttt{Controller} = By^{C} \qquad \texttt{Plant} = Cy^{AB} \qquad \texttt{System} = Cy^{A}$$

The wiring diagram represents a lens $\texttt{Controller} \otimes \texttt{Plant} \to \texttt{System}$.

$$By^{C} \otimes Cy^{AB} \longrightarrow Cy^{A}$$

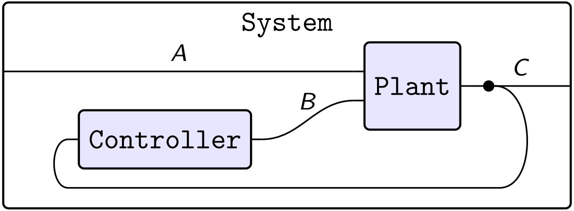
Moore machines and wiring diagrams as lenses



To summarize what we've said so far:

- A wiring diagram (WD) is a lens, e.g. $By^{C} \otimes Cy^{AB} \longrightarrow Cy^{A}$.
- Each Moore machine is a lens, e.g. $Sy^{S} \to By^{C}$ and $Ty^{T} \to Cy^{AB}$.

Moore machines and wiring diagrams as lenses



To summarize what we've said so far:

- A wiring diagram (WD) is a lens, e.g. $By^C \otimes Cy^{AB} \longrightarrow Cy^A$.
- Each Moore machine is a lens, e.g. $Sy^S \to By^C$ and $Ty^T \to Cy^{AB}$.

We can tensor the Moore machines and compose to obtain $STy^{ST} \to Cy^A$.

- So a wiring diagram is a formula for combining Moore machines.
- The whole story is lenses, through and through.

Poly and mode-dependent dynamics

All of the above on Moore machines and WDs took place in $\mathbf{Poly}_{monomial}$.

- An arbitrary polynomial p is a sum of monomials.
- A Moore machine is a map $Sy^S \to By^A$.
- Generalized Moore machine: a map $Sy^S \to p$ with $p \in \mathbf{Poly}$ arbitrary.
- We can think of this as *mode-dependence*:
    - The position—and hence directions—depends on the state.
    - Roughly, the input-output type can change based on state.

Poly and mode-dependent dynamics

All of the above on Moore machines and WDs took place in $\mathbf{Poly}_{monomial}$.

- An arbitrary polynomial p is a sum of monomials.
- A Moore machine is a map $Sy^S \to By^A$.
- Generalized Moore machine: a map $Sy^S \to p$ with $p \in \mathbf{Poly}$ arbitrary.
- We can think of this as *mode-dependence*:
  - The position—and hence directions—depends on the state.
  - Roughly, the input-output type can change based on state.

I'll discuss mode dependence for wiring diagrams by example.

Example

This whole picture represents one morphism in **Poly**.

- Let's suppose the company chooses who it wires to; this is its mode.
- Then both suppliers have interface wy.
- Company interface is $2y^w$: two modes, each of which is w-input.
- The outer box is just y, i.e. a closed system.

So the picture represents a map $wy \otimes wy \otimes 2y^w \to y$.

- That's a map $2w^2y^w \to y$.
- Equivalently, it's a function $2w^2 \to w$. Take it to be evaluation.
- In other words, the company's choice determines which w it receives.

Comonoids in $(\mathbf{Poly}, \circ, y)$

For Moore machines—usual or generalized—what makes $Sy^S \to p$ tick?

- We wrote some recursive formula for the "stream transducer".
- But it turns out that what we were seeing is really about comonoids.
- Comonoid: $k \in \mathbf{Poly}$, $\quad \delta\colon k \to k \circ k$, $\quad \epsilon\colon k \to y$, $\quad$ usual laws.
- A comonoid in $(\mathbf{Poly}, \circ, y)$ could be called a *polynomial comonad*.
- Sy^S has the structure of a comonad, the "store comonad".

Comonoids in $(\mathbf{Poly}, \circ, y)$

For Moore machines—usual or generalized—what makes $Sy^S \to p$ tick?

- We wrote some recursive formula for the "stream transducer".
- But it turns out that what we were seeing is really about comonoids.
- Comonoid: $k \in \mathbf{Poly}$, $\delta\colon k \to k \circ k$, $\epsilon\colon k \to y$, usual laws.
- A comonoid in $(\mathbf{Poly}, \circ, y)$ could be called a *polynomial comonad*.
- Sy^S has the structure of a comonad, the "store comonad".

How does it work?

Cofree comonoids and terminal coalgebras

The forgetful functor **Comon**(**Poly**) $\to$ **Poly** has a right adjoint, Cofree.

- Let $\mathscr{K} = (k, \delta, \epsilon)$ be a comonoid in $(\mathbf{Poly}, \circ, y)$.
- Given poly'l map $k \to p$, get a comonoid map $\mathscr{K} \to \mathrm{Cofree}(p)$.
- The formula for cofree comonoid on p in general is the limit:

$$1 \longleftarrow y{\cdot}p(1) \leftarrow y{\cdot}p(y{\cdot}p(1)) \leftarrow y{\cdot}p(y{\cdot}p(y{\cdot}p(1))) \leftarrow \cdots$$

- Substituting 1 for y we get the usual formula for terminal coalgebra.

$$1 \leftarrow p(1) \leftarrow p(p(1)) \leftarrow p(p(p(1))) \leftarrow \cdots$$

Cofree comonoids and terminal coalgebras

The forgetful functor **Comon**(**Poly**) $\to$ **Poly** has a right adjoint, Cofree.

- Let $\mathscr{K} = (k, \delta, \epsilon)$ be a comonoid in $(\mathbf{Poly}, \circ, y)$.
- Given poly'l map $k \to p$, get a comonoid map $\mathscr{K} \to \mathrm{Cofree}(p)$.
- The formula for cofree comonoid on p in general is the limit:

$$1 \longleftarrow y\cdot p(1) \leftarrow y\cdot p(y\cdot p(1)) \leftarrow y\cdot p(y\cdot p(y\cdot p(1))) \leftarrow \cdots$$

- Substituting 1 for y we get the usual formula for terminal coalgebra.

$$1 \leftarrow p(1) \leftarrow p(p(1)) \leftarrow p(p(p(1))) \leftarrow \cdots$$

Example:

- In the case $p = By^A$, we have $\mathrm{Cofree}(p) \cong B^{\mathrm{List}(A)}y^{\mathrm{List}(A)}$.
- So $Sy^S \to B^{\mathrm{List}(A)}y^{\mathrm{List}(A)}$ gives

$$S \times \mathrm{List}(A) \to B \quad \text{and} \quad S \times \mathrm{List}(A) \to S.$$

- Given the initial state s_0, we get back our stream transducer.

The amazing world of comonoids in Poly

Comonoids in **Poly** are amazing.

- Not only are they what make generalized Moore machines tick, but...
- Ahman-Uustalu showed that they're precisely categories!

$$\mathbf{Comon}(\mathbf{Poly}) \cong (\mathbf{Categories}, \mathbf{Cofunctors})$$

- An easy fact is that their coalgebras correspond to copresheaves!
- Garner showed that bimodules between them correspond to parametric right adjoints between these copresheaf categories!
- All this stuff is relevant for databases and data migration.
- It's also just shockingly cool from a theoretical perspective.

Outline

Summary

The category **Poly** is exceptionally rich.

- Four interacting monoidal structures, two closures, etc, etc.
- Comonoids $\mathscr{C}$ in $(\mathbf{Poly}, \circ, y)$ are categories.
- Discrete left $\mathscr{C}$-comodules are co-presheaves.
- Bimodules are parametric right adjoints.

Summary

The category **Poly** is exceptionally rich.

- Four interacting monoidal structures, two closures, etc, etc.
- Comonoids $\mathscr{C}$ in $(\mathbf{Poly}, \circ, y)$ are categories.
- Discrete left $\mathscr{C}$-comodules are co-presheaves.
- Bimodules are parametric right adjoints.

Poly can be used in several different applications.

- Containers in functional programming.
- Generalized lenses (as polynomials generalize monomials).
- Mode-dependent dynamical systems and wiring diagrams.
- Databases and data migration.

Poly provides an expressive notation and calculus for dynamics and data.

Thanks; comments and questions welcome!