Poly is an unreasonably effective abstraction; Might it relate to healthy systems?

David I. Spivak

TOPOS INSTITUTE

Finding the Right Abstractions for Healthy Systems
2023 January 08

Outline

1. **Introduction**
   - Why am I here?
   - Sense-making
   - Accounting systems
   - Interfaces and interaction
   - Plan for the talk
2. Introduction to Poly
3. Unreasonable effectiveness of Poly
4. What is health?
5. Conclusion

Why am I here?

For reasons I don't fully understand, math is powerful.

- Math is at the core of science and technology.
- In whatever sense math *tracks* reality, it empowers *prediction*.
- We can say earlier something about what *will* happen later.
- We can correctly track that our existence *will* catalyze some change.

Why am I here?

For reasons I don't fully understand, math is powerful.

- Math is at the core of science and technology.
- In whatever sense math *tracks* reality, it empowers *prediction*.
- We can say earlier something about what *will* happen later.
- We can correctly track that our existence *will* catalyze some change.

I'm here to participate in an endeavor to track what contributes to health.

- I want our world to be healthier, but what does that mean?
- We don't have the right abstractions for thinking about this clearly.
- E.g. should we distinguish between health, wealth, and wisdom?

Why am I here?

For reasons I don't fully understand, math is powerful.

- Math is at the core of science and technology.
- In whatever sense math *tracks* reality, it empowers *prediction*.
- We can say earlier something about what *will* happen later.
- We can correctly track that our existence *will* catalyze some change.

I'm here to participate in an endeavor to track what contributes to health.

- I want our world to be healthier, but what does that mean?
- We don't have the right abstractions for thinking about this clearly.
- E.g. should we distinguish between health, wealth, and wisdom?

The etymology of *health* points to a root of *wholeness*.

- This points us to the health of collectives rather than individuals.
- I am constituted by a collective, and I help constitute collectives.
- My health is measured by the extent to which both are *whole*.
- The question of wholeness is that of *integrity*: does it hold together?

Why am I here?

For reasons I don't fully understand, math is powerful.

- Math is at the core of science and technology.
- In whatever sense math *tracks* reality, it empowers *prediction*.
- We can say earlier something about what *will* happen later.
- We can correctly track that our existence *will* catalyze some change.

I'm here to participate in an endeavor to track what contributes to health.

- I want our world to be healthier, but what does that mean?
- We don't have the right abstractions for thinking about this clearly.
- E.g. should we distinguish between health, wealth, and wisdom?

The etymology of *health* points to a root of *wholeness*.

- This points us to the health of collectives rather than individuals.
- I am constituted by a collective, and I help constitute collectives.
- My health is measured by the extent to which both are *whole*.
- The question of wholeness is that of *integrity*: does it hold together?

I'm here to think with you about math for helping systems be healthier.

Sense-making

What enables a collective to act in concert?

- What bring coherence, cooperation, coordination, integrity?
- What prevents friction, loss, stepping on toes, thwarting ourselves?
- Different parts of the collective see from different perspectives.
- Each acts according to their own mandate. What brings coordination?

Sense-making

What enables a collective to act in concert?

- What bring coherence, cooperation, coordination, integrity?
- What prevents friction, loss, stepping on toes, thwarting ourselves?
- Different parts of the collective see from different perspectives.
- Each acts according to their own mandate. What brings coordination?

I think the word *sense* is appropriate.

- By *sense* I don't mean raw perception.
- Think Spidey-sense, sense of direction, of danger, of humor.
- Very complex situations are boiled down into something actionable.
- We talk about *making* sense; is this a pun, or can sense be made?

Sense-making

What enables a collective to act in concert?

- What bring coherence, cooperation, coordination, integrity?
- What prevents friction, loss, stepping on toes, thwarting ourselves?
- Different parts of the collective see from different perspectives.
- Each acts according to their own mandate. What brings coordination?

I think the word *sense* is appropriate.

- By *sense* I don't mean raw perception.
- Think Spidey-sense, sense of direction, of danger, of humor.
- Very complex situations are boiled down into something actionable.
- We talk about *making* sense; is this a pun, or can sense be made?

Consider a snapshot of two math students, both wanting to succeed:

- Student A is faithfully copying down what the teacher says.
- Student B argues, points, gestures, explains, and then... gets it!
- The collective of sense-makers in B found a way to align.
- Having "made sense", student B performs better.

Working language

As a collective, we want to get a better sense of what health is.

- We may need to argue about it, work with it, gesture at it.
- In the end, we want to align our senses of health.
- Why does this entail “finding the right abstractions”?

Working language

As a collective, we want to get a better sense of what health is.

- We may need to argue about it, work with it, gesture at it.
- In the end, we want to align our senses of health.
- Why does this entail "finding the right abstractions"?

Language *works* in the sense of basic physics.

- Language like "pass the salt" can direct displacement of objects.
- DNA as a 4-letter language coding for amino acids also *works*.
- Computer programming languages direct changes in voltage levels.
- Lang. is selected for expressivity, comm've success, processing ease.

Working language

As a collective, we want to get a better sense of what health is.

- We may need to argue about it, work with it, gesture at it.
- In the end, we want to align our senses of health.
- Why does this entail "finding the right abstractions"?

Language *works* in the sense of basic physics.

- Language like "pass the salt" can direct displacement of objects.
- DNA as a 4-letter language coding for amino acids also *works*.
- Computer programming languages direct changes in voltage levels.
- Lang. is selected for expressivity, comm've success, processing ease.

Language is$^?$ fast-moving pattern that can be unpacked in local context.

- I send you patterned air (pressure waves) about wanting salt.
- Coordinated signals (patterned neural) are sent around your brain.
- Each makes sense in the local context and the results cohere.
- Disputes between components can be resolved through language.

Working language

As a collective, we want to get a better sense of what health is.

- We may need to argue about it, work with it, gesture at it.
- In the end, we want to align our senses of health.
- Why does this entail "finding the right abstractions"?

Language *works* in the sense of basic physics.

- Language like "pass the salt" can direct displacement of objects.
- DNA as a 4-letter language coding for amino acids also *works*.
- Computer programming languages direct changes in voltage levels.
- Lang. is selected for expressivity, comm've success, processing ease.

Language is? fast-moving pattern that can be unpacked in local context.

- I send you patterned air (pressure waves) about wanting salt.
- Coordinated signals (patterned neural) are sent around your brain.
- Each makes sense in the local context and the results cohere.
- Disputes between components can be resolved through language.

I would guess that any healthy collective relies on appropriate language.

Mathematical fields as accounting systems

What does it mean that components resolve disputes through language?

- Each body part, employee in an org, member of a workshop...
- ...has its own view of the situation and set of commitments to honor.
- Accountability means being able to explain actions in collective terms.
- Signals show up on our interface, allow others to coordinate with us.

Mathematical fields as accounting systems

What does it mean that components resolve disputes through language?

- Each body part, employee in an org, member of a workshop...
- ...has its own view of the situation and set of commitments to honor.
- Accountability means being able to explain actions in collective terms.
- Signals show up on our interface, allow others to coordinate with us.

I think of mathematical fields as crystalized accounting systems.

- Arithmetic accounts for the flow of quantities, as in finance.
- Hilbert spaces account for the states of elementary particles, as in QM.
- Probability distributions account for likelihoods, as in game theory.
- Each is crystalized: hardened and coherent. Think laws of arithmetic.

Mathematical fields as accounting systems

What does it mean that components resolve disputes through language?

- Each body part, employee in an org, member of a workshop...
- ...has its own view of the situation and set of commitments to honor.
- Accountability means being able to explain actions in collective terms.
- Signals show up on our interface, allow others to coordinate with us.

I think of mathematical fields as crystalized accounting systems.

- Arithmetic accounts for the flow of quantities, as in finance.
- Hilbert spaces account for the states of elementary particles, as in QM.
- Probability distributions account for likelihoods, as in game theory.
- Each is crystalized: hardened and coherent. Think laws of arithmetic.

Math is a high-fidelity language for systematic tracking.

- The language provides a conceptual overlay for the phenomena.
- The rules let us regulate each other: check each others' work.
- The regularity lets the collective share accounts w/o interpretive loss.

Mathematical fields as accounting systems

What does it mean that components resolve disputes through language?

- Each body part, employee in an org, member of a workshop...
- ...has its own view of the situation and set of commitments to honor.
- Accountability means being able to explain actions in collective terms.
- Signals show up on our interface, allow others to coordinate with us.

I think of mathematical fields as crystalized accounting systems.

- Arithmetic accounts for the flow of quantities, as in finance.
- Hilbert spaces account for the states of elementary particles, as in QM.
- Probability distributions account for likelihoods, as in game theory.
- Each is crystalized: hardened and coherent. Think laws of arithmetic.

Math is a high-fidelity language for systematic tracking.

- The language provides a conceptual overlay for the phenomena.
- The rules let us regulate each other: check each others' work.
- The regularity lets the collective share accounts w/o interpretive loss.

So math might help heal a system by improving communication within it.

Interfaces and interaction

We want math for improving communication within collectives.

- I'm not sure how far this goes toward health, wealth, wisdom.
- But it's a nice fixed point: we can aim for it *here and now*.

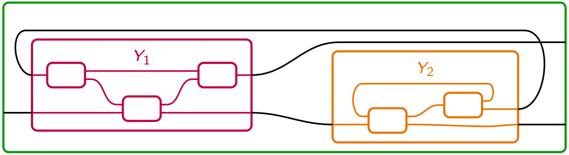
Interfaces and interaction

We want math for improving communication within collectives.

- I'm not sure how far this goes toward health, wealth, wisdom.
- But it's a nice fixed point: we can aim for it *here and now*.

I want to think of interfaces, rather than entities.

- Interfaces—aka surfaces, boundaries, membranes—are screens.
- Inside or outside the interface may be a whole collective.
- But each is "private", not directly knowable on the other side.
- The interface is just telling us the "input-output language".



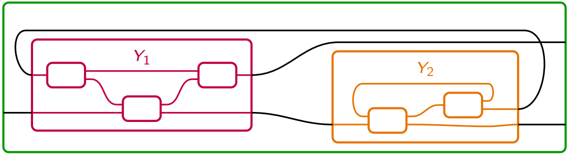
Interfaces and interaction

We want math for improving communication within collectives.

- I'm not sure how far this goes toward health, wealth, wisdom.
- But it's a nice fixed point: we can aim for it *here and now*.

I want to think of interfaces, rather than entities.

- Interfaces—aka surfaces, boundaries, membranes—are screens.
- Inside or outside the interface may be a whole collective.
- But each is "private", not directly knowable on the other side.
- The interface is just telling us the "input-output language".



I want to present the category **Poly** and make the case for its use.

- I claim it's an unreasonably effective language for computer science.
- The main object of study can be understood as an interface.

Plan for the talk

During the remainder of the talk, I will:

- Give an intuitive mathematical introduction to **Poly**,
- Explain why I think it's unreasonably effective in computer science,
- Consider whether/how it help with our subject matter (health), and
- Conclude with a summary.

Outline

1 Introduction

2 **Introduction to Poly**
- Definition and intuition
- Lenses, Moore machines, and Mealy machines

3 Unreasonable effectiveness of Poly

4 What is health?

5 Conclusion

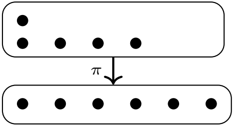
Definition and intuition

A *polynomial* p is essentially a data structure. Here are three viewpoints:

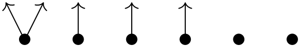
Algebraic

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

Bundle



Corolla forest



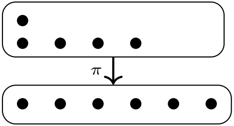
Definition and intuition

A *polynomial* p is essentially a data structure. Here are three viewpoints:

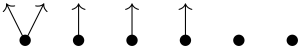
Algebraic

$y^2 + 3y + 2$

Bundle



Corolla forest



Cat. description: **Poly** = "sums of representables functors **Set** → **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.
- Note that $I = p(1)$; this is a convenient fact. Write $p[i]$ for P_i.
- (We can use many other categories in place of **Set**, but let's not.)

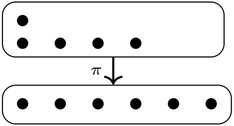
Definition and intuition

A *polynomial* p is essentially a data structure. Here are three viewpoints:

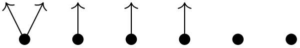
Algebraic

$y^2 + 3y + 2$

Bundle



Corolla forest



Cat. description: **Poly** = "sums of representables functors **Set** → **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.
- Note that $I = p(1)$; this is a convenient fact. Write $p[i]$ for P_i.
- (We can use many other categories in place of **Set**, but let's not.)

Other ways to see a polynomial $p = \sum_{i \in I} y^{p[i]}$ as an interface:

- A set I of *types*; each type $i : I$ has a set $p[i]$ of *terms*.
- A set I of *problems*; each problem $i : I$ has a set $p[i]$ of *solutions*.
- A set I of *body positions*; each pos'n $i : I$ has a set $p[i]$ of *sensations*

Operations: $+, \times, \otimes, \triangleleft, [-,-], \left[\frac{-}{-}\right]$

Given two interfaces p, q, there are many ways to get another interface.

- For each we'll say the problems and solutions for resulting interface.
- Sum $p+q$: problem is $i \in p(1)$ or $j \in q(1)$; solve it.

Operations: $+, \times, \otimes, \triangleleft, [-,-], \begin{bmatrix}-\\-\end{bmatrix}$

Given two interfaces p, q, there are many ways to get another interface.

- For each we'll say the problems and solutions for resulting interface.
- Sum $p + q$: problem is $i \in p(1)$ or $j \in q(1)$; solve it.
- Product $p \times q$: problem is pair $(i, j) \in p(1) \times q(1)$; solve either.

Operations: $+, \times, \otimes, \triangleleft, [-,-], \left[\frac{-}{-}\right]$

Given two interfaces p, q, there are many ways to get another interface.

- For each we'll say the problems and solutions for resulting interface.
- Sum $p + q$: problem is $i \in p(1)$ or $j \in q(1)$; solve it.
- Product $p \times q$: problem is pair $(i, j) \in p(1) \times q(1)$; solve either.
- Dirichlet product $p \otimes q$: prob'm is pair $(i, j) \in p(1) \times q(1)$; solve both.

Operations: $+, \times, \otimes, \triangleleft, [-,-], \left[\begin{smallmatrix}-\\-\end{smallmatrix}\right]$

Given two interfaces p, q, there are many ways to get another interface.

- For each we'll say the problems and solutions for resulting interface.
- Sum $p + q$: problem is $i \in p(1)$ or $j \in q(1)$; solve it.
- Product $p \times q$: problem is pair $(i, j) \in p(1) \times q(1)$; solve either.
- Dirichlet product $p \otimes q$: prob'm is pair $(i, j) \in p(1) \times q(1)$; solve both.
- Substitution product $p \triangleleft q$: prob'm is choice of $i \in p(1)$ and...
- ...for every solution a problem $j \in q(1)$; solve first then second.

Operations: $+, \times, \otimes, \triangleleft, [-,-], \left[\begin{smallmatrix}-\\-\end{smallmatrix}\right]$

Given two interfaces p, q, there are many ways to get another interface.

- For each we'll say the problems and solutions for resulting interface.
- Sum $p + q$: problem is $i \in p(1)$ or $j \in q(1)$; solve it.
- Product $p \times q$: problem is pair $(i, j) \in p(1) \times q(1)$; solve either.
- Dirichlet product $p \otimes q$: prob'm is pair $(i, j) \in p(1) \times q(1)$; solve both.
- Substitution product $p \triangleleft q$: prob'm is choice of $i \in p(1)$ and...
- ...for every solution a problem $j \in q(1)$; solve first then second.
- Internal hom $[p, q]$: problem is polynomial map $\varphi\colon p \to q$;...
- ...soln: problem $i \in p(1)$ and solution to its image $\varphi_1(i) \in q(1)$.

Operations: $+, \times, \otimes, \triangleleft, [-,-], \left[\frac{-}{-}\right]$

Given two interfaces p, q, there are many ways to get another interface.

- For each we'll say the problems and solutions for resulting interface.
- Sum $p + q$: problem is $i \in p(1)$ or $j \in q(1)$; solve it.
- Product $p \times q$: problem is pair $(i, j) \in p(1) \times q(1)$; solve either.
- Dirichlet product $p \otimes q$: prob'm is pair $(i, j) \in p(1) \times q(1)$; solve both.
- Substitution product $p \triangleleft q$: prob'm is choice of $i \in p(1)$ and...
- ...for every solution a problem $j \in q(1)$; solve first then second.
- Internal hom $[p, q]$: problem is polynomial map $\varphi \colon p \to q$;...
- ...soln: problem $i \in p(1)$ and solution to its image $\varphi_1(i) \in q(1)$.
- The last one is "Left Kan extension"; next slide.

Operations: $+, \times, \otimes, \triangleleft, [-,-], \left[\frac{-}{-}\right]$

Given two interfaces p, q, there are many ways to get another interface.

- For each we'll say the problems and solutions for resulting interface.
- Sum $p+q$: problem is $i \in p(1)$ or $j \in q(1)$; solve it.
- Product $p \times q$: problem is pair $(i,j) \in p(1) \times q(1)$; solve either.
- Dirichlet product $p \otimes q$: prob'm is pair $(i,j) \in p(1) \times q(1)$; solve both.
- Substitution product $p \triangleleft q$: prob'm is choice of $i \in p(1)$ and...
- ...for every solution a problem $j \in q(1)$; solve first then second.
- Internal hom $[p,q]$: problem is polynomial map $\varphi \colon p \to q$;...
- ...soln: problem $i \in p(1)$ and solution to its image $\varphi_1(i) \in q(1)$.
- The last one is "Left Kan extension"; next slide.

Letting $p \coloneqq \sum_{i \in p(1)} y^{p_i}$ and $q \coloneqq \sum_{j \in q(1)} y^{q_j}$

$$p \times q = \sum_{(i,j)} y^{p[i]+q[j]} \qquad p \otimes q = \sum_{(i,j)} y^{p[i] \times q[j]}$$

$$p \triangleleft q = \sum_{i \in p(1)} \sum_{j \colon p[i] \to q(1)} y^{\sum_{x \in p[i]} q[jx]} \qquad [p,q] = \sum_{\varphi \colon p \to q} y^{\sum_{i \in p(1)} q[\varphi_1 i]}$$

Comonoids are categories

Poly has a lot of amazing surprises, as we'll see. Here's one.

- The substitution product $p \triangleleft q$ means plug q into p.
- So $y^2 \triangleleft (y+1) \cong y^2 + 2y + 1$. Not symmetric! $(y+1) \triangleleft y^2 = y^2 + 1$.
- But it's a monoidal structure. The unit is y because $y \triangleleft p = p = p \triangleleft y$.

Comonoids are categories

Poly has a lot of amazing surprises, as we'll see. Here's one.

- The substitution product $p \triangleleft q$ means plug q into p.
- So $y^2 \triangleleft (y+1) \cong y^2 + 2y + 1$. Not symmetric! $(y+1) \triangleleft y^2 = y^2 + 1$.
- But it's a monoidal structure. The unit is y because $y \triangleleft p = p = p \triangleleft y$.

In any mon'l cat'y, it's interesting to consider the monoids and comonoids.

- In the case of $(\mathbf{Poly}, y, \triangleleft)$, the comonoids are exactly categories!
- If $\mathcal{C}$ is a category, for any $c \in \mathrm{Ob}(\mathcal{C})$ define $\mathcal{C}[c] := \sum_{c' \in \mathrm{Ob}(\mathcal{C})} \mathcal{C}(c, c')$.

Comonoids are categories

Poly has a lot of amazing surprises, as we'll see. Here's one.

- The substitution product $p \triangleleft q$ means plug q into p.
- So $y^2 \triangleleft (y+1) \cong y^2 + 2y + 1$. Not symmetric! $(y+1) \triangleleft y^2 = y^2 + 1$.
- But it's a monoidal structure. The unit is y because $y \triangleleft p = p = p \triangleleft y$.

In any mon'l cat'y, it's interesting to consider the monoids and comonoids.

- In the case of $(\mathbf{Poly}, y, \triangleleft)$, the comonoids are exactly categories!
- If $\mathcal{C}$ is a category, for any $c \in \mathrm{Ob}(\mathcal{C})$ define $\mathcal{C}[c] := \sum_{c' \in \mathrm{Ob}(\mathcal{C})} \mathcal{C}(c, c')$.
- Then the associated polynomial is $p_{\mathcal{C}} := \sum_{c \in \mathrm{Ob}(\mathcal{C})} y^{\mathcal{C}[c]}$.
- Identities, codomains, and compositions are given by coherent maps

$$\epsilon \colon p_c \to y \qquad \text{and} \qquad \delta \colon p_c \to p_c \triangleleft p_c$$

Comonoids are categories

Poly has a lot of amazing surprises, as we'll see. Here's one.

- The substitution product $p \triangleleft q$ means plug q into p.
- So $y^2 \triangleleft (y+1) \cong y^2 + 2y + 1$. Not symmetric! $(y+1) \triangleleft y^2 = y^2 + 1$.
- But it's a monoidal structure. The unit is y because $y \triangleleft p = p = p \triangleleft y$.

In any mon'l cat'y, it's interesting to consider the monoids and comonoids.

- In the case of $(\mathbf{Poly}, y, \triangleleft)$, the comonoids are exactly categories!
- If $\mathcal{C}$ is a category, for any $c \in \mathrm{Ob}(\mathcal{C})$ define $\mathcal{C}[c] := \sum_{c' \in \mathrm{Ob}(\mathcal{C})} \mathcal{C}(c, c')$.
- Then the associated polynomial is $p_{\mathcal{C}} := \sum_{c \in \mathrm{Ob}(\mathcal{C})} y^{\mathcal{C}[c]}$.
- Identities, codomains, and compositions are given by coherent maps

$$\epsilon \colon p_c \to y \qquad \text{and} \qquad \delta \colon p_c \to p_c \triangleleft p_c$$

All that to say that comonoids in **Poly** are exactly categories!

- Maps between comonoids are not functors; they're "cofunctors".
- Ex: every Sy^S has a comonoid structure. Lawful lenses = cofunctors.
- Denote the category of categories and cofunctors by $\mathbf{Cat}^\sharp$.

Lenses, Moore machines, and Mealy machines

For any p, q as above, we have $\begin{bmatrix} q \\ p \end{bmatrix} = \sum_{i \in p(1)} y^{q(p[i])}$.

- In particular, we can regard $A, B \in \mathbf{Set}$ as constant polynomials.
- Then $\begin{bmatrix} A \\ B \end{bmatrix} = By^A$. Maps between these are "lenses".
- A map $\begin{bmatrix} A \\ B \end{bmatrix} \to \begin{bmatrix} A' \\ B' \end{bmatrix}$ is a natural transf'n $By^A \to B'y^{A'}$. It consists of
  - get: $B \to B'$
  - put: $B \times A' \to A$
  - These come up in functional programming.

Lenses, Moore machines, and Mealy machines

For any p, q as above, we have $\begin{bmatrix} q \\ p \end{bmatrix} = \sum_{i \in p(1)} y^{q(p[i])}$.

- In particular, we can regard $A, B \in \mathbf{Set}$ as constant polynomials.
- Then $\begin{bmatrix} A \\ B \end{bmatrix} = By^A$. Maps between these are "lenses".
- A map $\begin{bmatrix} A \\ B \end{bmatrix} \to \begin{bmatrix} A' \\ B' \end{bmatrix}$ is a natural transf'n $By^A \to B'y^{A'}$. It consists of
  - get: $B \to B'$
  - put: $B \times A' \to A$
  - These come up in functional programming.

What's the point? The main math definition stuff is done. Let's get to it.

- A map $\begin{bmatrix} S \\ S \end{bmatrix} \to \begin{bmatrix} A \\ B \end{bmatrix}$ is a *Moore machine*. It consists of:
- State set S, a readout $f^{\text{rdt}} \colon S \to B$, and dynamics $f^{\text{dyn}} \colon S \times A \to S$.

Lenses, Moore machines, and Mealy machines

For any p, q as above, we have $\begin{bmatrix} q \\ p \end{bmatrix} = \sum_{i \in p(1)} y^{q(p[i])}$.

- In particular, we can regard $A, B \in \mathbf{Set}$ as constant polynomials.
- Then $\begin{bmatrix} A \\ B \end{bmatrix} = By^A$. Maps between these are "lenses".
- A map $\begin{bmatrix} A \\ B \end{bmatrix} \to \begin{bmatrix} A' \\ B' \end{bmatrix}$ is a natural transf'n $By^A \to B'y^{A'}$. It consists of
  - get: $B \to B'$
  - put: $B \times A' \to A$
  - These come up in functional programming.

What's the point? The main math definition stuff is done. Let's get to it.

- A map $\begin{bmatrix} S \\ S \end{bmatrix} \to \begin{bmatrix} A \\ B \end{bmatrix}$ is a *Moore machine*. It consists of:
- State set S, a readout $f^{\text{rdt}} \colon S \to B$, and dynamics $f^{\text{dyn}} \colon S \times A \to S$.
- Given some initial $s_0 \in S$ and an input list $a_0, \ldots, a_n$, let...
- ...$s_{i+1} \coloneqq f^{\text{dyn}}(s_i, a_i)$ and $b_i \coloneqq f^{\text{rdt}}(s_i)$. Get output list $b_0, \ldots, b_n$.

Lenses, Moore machines, and Mealy machines

For any p, q as above, we have $\begin{bmatrix} q \\ p \end{bmatrix} = \sum_{i \in p(1)} y^{q(p[i])}$.

- In particular, we can regard $A, B \in \mathbf{Set}$ as constant polynomials.
- Then $\begin{bmatrix} A \\ B \end{bmatrix} = By^A$. Maps between these are "lenses".
- A map $\begin{bmatrix} A \\ B \end{bmatrix} \to \begin{bmatrix} A' \\ B' \end{bmatrix}$ is a natural transf'n $By^A \to B'y^{A'}$. It consists of
  - get: $B \to B'$
  - put: $B \times A' \to A$
  - These come up in functional programming.

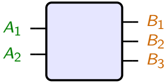
What's the point? The main math definition stuff is done. Let's get to it.

- A map $\begin{bmatrix} S \\ S \end{bmatrix} \to \begin{bmatrix} A \\ B \end{bmatrix}$ is a *Moore machine*. It consists of:
- State set S, a readout $f^{\mathrm{rdt}} \colon S \to B$, and dynamics $f^{\mathrm{dyn}} \colon S \times A \to S$.
- Given some initial $s_0 \in S$ and an input list $a_0, \ldots, a_n$, let...
- ...$s_{i+1} := f^{\mathrm{dyn}}(s_i, a_i)$ and $b_i := f^{\mathrm{rdt}}(s_i)$. Get output list $b_0, \ldots, b_n$.
- A map $\begin{bmatrix} S \\ S \end{bmatrix} \to [Ay, By]$ is a *Mealy machine.*
- It consists of state set S and a function $S \times A \to S \times B$.
- Again, it can transform a list of inputs into a list of outputs.

Depicting Moore machine interfaces

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

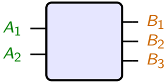
If, e.g. $A = A_1 \times A_2$ and $B = B_1 \times B_2 \times B_3$, we will instead draw:



Depicting Moore machine interfaces

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

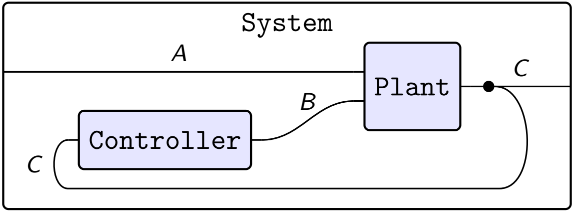
If, e.g. $A = A_1 \times A_2$ and $B = B_1 \times B_2 \times B_3$, we will instead draw:



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:

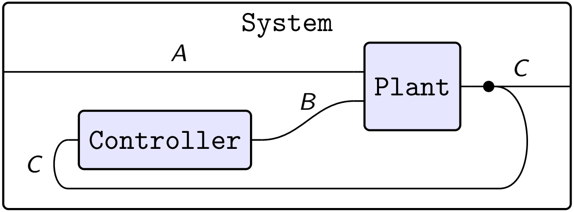


It includes three interfaces: `Controller`, `Plant`, and `System`.

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

Wiring diagrams

Here's a picture of a wiring diagram:



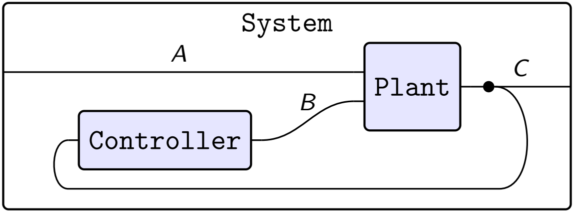
It includes three interfaces: Controller, Plant, and System.

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

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

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

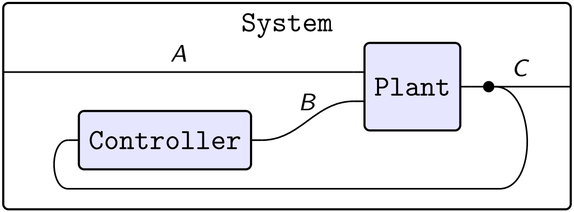
Moore machines and wiring diagrams as lenses



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 (monomials), through and through.
- For "mode dependence" where interfaces can change, use gen'l polys.

Outline

1. Introduction

2. Introduction to Poly

3. **Unreasonable effectiveness of Poly**
   - Category theory in Computer Science
   - Functional programming
   - Databases and data migration
   - Dependent type theory
   - Effects handlers
   - Dynamic organizational systems
   - Synthetic programming
   - The “scientist category”
   - Mathematical properties of **Poly**

Category theory in Computer Science

Category theory has been useful in computer science.

- Functional programming languages, e.g. Haskell.
- Here, types are objects, programs are morphisms.
- The result is a cartesian closed category: tupling and function types.
- Side effects are handled by monads.

Category theory in Computer Science

Category theory has been useful in computer science.

- Functional programming languages, e.g. Haskell.
- Here, types are objects, programs are morphisms.
- The result is a cartesian closed category: tupling and function types.
- Side effects are handled by monads.

Poly can add a lot to this story.

- First, note that it's already involved in many ways.
  - Algebraic data types are free monads on polynomial functors.
  - Initial algebras and final coalgebras for poly's are very common.
  - Lenses are maps between monomials.
- But we will see that **Poly** goes far beyond functional programming.
- We've seen it's relevant for finite state (Moore) machines. Also:
  - Databases and data migration,
  - Dependent type theory,
  - Effects handling,
  - Rewriting workflows,
  - Deep learning

Functional programming

In functional languages such as Haskell, you often see things like this:

```
data Foo y = Bar y y y | Baz y y | Qux | Quux
data Maybe y = Just y | Nothing
```

- These are polynomials: $y^3 + y^2 + 2$ and $y + 1$ respectively.
- They're "polymorphic" in that
  - they act on any Haskell type `Y` in place of the variable `y`, and
  - for any map `f :  Y1 -> Y2` there's a map `Foo Y1 -> Foo Y2`

Functional programming

In functional languages such as Haskell, you often see things like this:

```
data Foo y = Bar y y y | Baz y y | Qux | Quux
data Maybe y = Just y | Nothing
```

- These are polynomials: $\mathtt{y}^3 + \mathtt{y}^2 + 2$ and $\mathtt{y} + 1$ respectively.
- They're "polymorphic" in that
  - they act on any Haskell type `Y` in place of the variable `y`, and
  - for any map `f : Y1 -> Y2` there's a map `Foo Y1 -> Foo Y2`

Another thing you see in Haskell is something like this:

```
List a = Nil | Cons a (List a)
```

...for some type a, e.g. `a = Int`. What is going on here?

- This the *algebraic data type* corresponding to $p_A := 1 + Ay$.

Functional programming

In functional languages such as Haskell, you often see things like this:

```
data Foo y = Bar y y y | Baz y y | Qux | Quux
data Maybe y = Just y | Nothing
```

- These are polynomials: $\mathtt{y}^3 + \mathtt{y}^2 + 2$ and $\mathtt{y} + 1$ respectively.
- They're "polymorphic" in that
  - they act on any Haskell type `Y` in place of the variable `y`, and
  - for any map `f :  Y1 -> Y2` there's a map `Foo Y1 -> Foo Y2`

Another thing you see in Haskell is something like this:

```
List a = Nil | Cons a (List a)
```

...for some type a, e.g. `a = Int`. What is going on here?

- This the *algebraic data type* corresponding to $p_A \coloneqq 1 + A\mathcal{y}$.
- Every polynomial has an initial algebra and final coalgebra.
- The initial algebra of p_A is carried by $\sum_{n \in \mathbb{N}} A^n$, classic lists.
- The terminal coalgebra of p_A is carried by $A^{\mathbb{N}} + \sum_{n \in \mathbb{N}} A^n$, streams.

Databases and data migration

Databases are used throughout computer science.

- A database consists of a *schema*, the things and how they relate,...
- ...and *data*, which are examples of the things and their relationships.
- A useful CT story for this: schema $=$ category, data $=$ functor to $\mathbf{Set}$.
- Data migration means moving data from one schema to another.
- The most useful: disjoint unions of conjunctive (duc-) queries.

Databases and data migration

Databases are used throughout computer science.

- A database consists of a *schema*, the things and how they relate,...
- ...and *data*, which are examples of the things and their relationships.
- A useful CT story for this: schema = category, data = functor to **Set**.
- Data migration means moving data from one schema to another.
- The most useful: disjoint unions of conjunctive (duc-) queries.

All of this has a beautiful story in terms of polynomial functors.

- Indeed, schema = category $\mathcal{C}$ = polynomial comonad (c, ϵ, δ).
- And data = functor $\mathcal{C} \to \mathbf{Set}$ = c-coalgebra.
- Data migrations from $\mathcal{C}$ to $\mathcal{D}$ are exactly (c, d)-bicomodules.

Databases and data migration

Databases are used throughout computer science.

- A database consists of a *schema*, the things and how they relate,...
- ...and *data*, which are examples of the things and their relationships.
- A useful CT story for this: schema = category, data = functor to **Set**.
- Data migration means moving data from one schema to another.
- The most useful: disjoint unions of conjunctive (duc-) queries.

All of this has a beautiful story in terms of polynomial functors.

- Indeed, schema = category $\mathcal{C}$ = polynomial comonad (c, ϵ, δ).
- And data = functor $\mathcal{C} \to \mathbf{Set}$ = c-coalgebra.
- Data migrations from $\mathcal{C}$ to $\mathcal{D}$ are exactly (c, d)-bicomodules.

Often databases are considered ugly, but the math here is cat'ly very clean.

Dependent type theory

Dependent types are what proof assistants like Coq&Lean are based on.

- Idea: a type can depend on values of another type.
- Eg: a category consists of a type O of objects and then...
- ...for every $o_1, o_2 : O$, a type $M(o_1, o_2)$ of morphisms and then...
- ...identities, compositions, rules, all depending on the previous stuff.

Dependent type theory

Dependent types are what proof assistants like Coq&Lean are based on.

- Idea: a type can depend on values of another type.
- Eg: a category consists of a type O of objects and then...
- ...for every $o_1, o_2 : O$, a type $M(o_1, o_2)$ of morphisms and then...
- ...identities, compositions, rules, all depending on the previous stuff.

Steve Awodey showed a tight connection between polynomials and DTT.

- He showed that a "natural model" of DTT is given by...
- ...a cartesian polynomial monad (m, η, μ) and a pseudo-algebra for it.
- The idea follows our conception of m as "types and terms".

Dependent type theory

Dependent types are what proof assistants like Coq&Lean are based on.

- Idea: a type can depend on values of another type.
- Eg: a category consists of a type O of objects and then...
- ...for every $o_1, o_2 : O$, a type $M(o_1, o_2)$ of morphisms and then...
- ...identities, compositions, rules, all depending on the previous stuff.

Steve Awodey showed a tight connection between polynomials and DTT.

- He showed that a "natural model" of DTT is given by...
- ...a cartesian polynomial monad (m, η, μ) and a pseudo-algebra for it.
- The idea follows our conception of m as "types and terms".
- A type in $m \triangleleft m$ is: a type in m and for every term, a type in m.
- The multiplication map $\mu\colon m \triangleleft m \to m$ realizes every such...
- ...compound type as a type in m. This tells you how to interpret Σ.
- He shows you can interpret Π-types using a m-pseudoalgebra.
- The type-forming and term-forming rules of DTT arise as the axioms.

Dependent type theory

Dependent types are what proof assistants like Coq&Lean are based on.

- Idea: a type can depend on values of another type.
- Eg: a category consists of a type O of objects and then...
- ...for every $o_1, o_2 : O$, a type $M(o_1, o_2)$ of morphisms and then...
- ...identities, compositions, rules, all depending on the previous stuff.

Steve Awodey showed a tight connection between polynomials and DTT.

- He showed that a “natural model” of DTT is given by...
- ...a cartesian polynomial monad (m, η, μ) and a pseudo-algebra for it.
- The idea follows our conception of m as “types and terms”.
- A type in $m \triangleleft m$ is: a type in m and for every term, a type in m.
- The multiplication map $\mu\colon m \triangleleft m \to m$ realizes every such...
- ...compound type as a type in m. This tells you how to interpret Σ.
- He shows you can interpret Π-types using a m-pseudoalgebra.
- The type-forming and term-forming rules of DTT arise as the axioms.

So the high-level language of proof assistants has semantics in **Poly**.

Effects handlers

In imperative programming, effects aren't unusual; they're the norm.

- The FP approach is to treat effects as scary, and use monads.
- But the result may be a pretty hairy mess of monads.
- Instead, one could put effects-handling as front and center.
- The issue is whether this still has nice denotational semantics.

Effects handlers

In imperative programming, effects aren't unusual; they're the norm.

- The FP approach is to treat effects as scary, and use monads.
- But the result may be a pretty hairy mess of monads.
- Instead, one could put effects-handling as front and center.
- The issue is whether this still has nice denotational semantics.

Owen Lynch and I designed a language for effects handling using **Poly**.

- Say the REPL has interface $q \coloneqq \mathtt{String}\, y + y^{\mathtt{String}} + y$
- Say the "CPU" has interface $p \coloneqq \sum_{r,r':\mathtt{Reg}} \{\mathtt{ADD}, \mathtt{MUL}, \mathtt{OUT}\}y + \mathtt{BEQ}y^2$
- The effects handler consists of a polynomial m and a map

  $$m \triangleleft p \to q \triangleleft m$$

  with states $m(1)$ and m_i-many threads for each state $i : m(1)$.
- This φ lets the REPL run the CPU.
- Effects handlers give rise to bicomodules (data migrators).

Effects handlers

In imperative programming, effects aren't unusual; they're the norm.

- The FP approach is to treat effects as scary, and use monads.
- But the result may be a pretty hairy mess of monads.
- Instead, one could put effects-handling as front and center.
- The issue is whether this still has nice denotational semantics.

Owen Lynch and I designed a language for effects handling using **Poly**.

- Say the REPL has interface $q \coloneqq \mathtt{String}\, y + y^{\mathtt{String}} + y$
- Say the "CPU" has interface $p \coloneqq \sum_{r,r':\mathtt{Reg}} \{\mathtt{ADD}, \mathtt{MUL}, \mathtt{OUT}\}y + \mathtt{BEQ}y^2$
- The effects handler consists of a polynomial m and a map

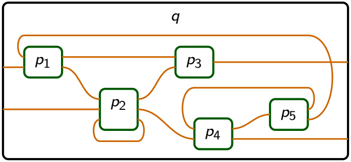
$$m \triangleleft p \to q \triangleleft m$$

  with states $m(1)$ and m_i-many threads for each state $i : m(1)$.
- This φ lets the REPL run the CPU.
- Effects handlers give rise to bicomodules (data migrators).

These are cat'ly nice: they form a distributive-monoidal double category.

Dynamic organizational systems

Earlier we discussed wiring diagrams interconnecting dynamical systems.

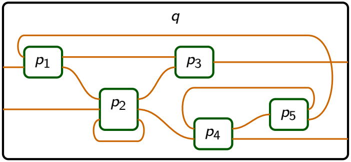


This implies that the interaction pattern is fixed for all time.

- Instead, we might want the wiring diagram to be able to change.
- It should change based on what flows through the wires.

Dynamic organizational systems

Earlier we discussed wiring diagrams interconnecting dynamical systems.



This implies that the interaction pattern is fixed for all time.

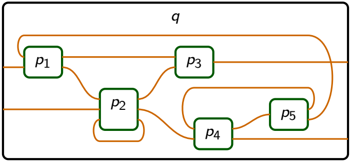
- Instead, we might want the wiring diagram to be able to change.
- It should change based on what flows through the wires.

Luckily, this is exactly the semantics of a $[p_1 \otimes \cdots \otimes p_5, q]$-machine.

- A state of the machine would output an interaction pattern $\varphi\colon p \to q$.
- And it would take as input an output of each p_i and an input to q.

Dynamic organizational systems

Earlier we discussed wiring diagrams interconnecting dynamical systems.



This implies that the interaction pattern is fixed for all time.

- Instead, we might want the wiring diagram to be able to change.
- It should change based on what flows through the wires.

Luckily, this is exactly the semantics of a $[p_1 \otimes \cdots \otimes p_5, q]$-machine.

- A state of the machine would output an interaction pattern $\varphi\colon p \to q$.
- And it would take as input an output of each p_i and an input to q.

In certain cases this can be done coherently: *dynamic operads*.

- Deep learning, prediction markets, Hebbian learning, open games.
- The way subordinates aggregate info changes based on what flows.

Synthetic programming

In synthetic programming, you have a type and you want a term of it.

- For example, you want a term of type $A \to (B \times C)$.
- The interface for a synthetic programmer can be given by $p : \mathbf{Poly}$.
- This is the "problems and solutions" semantics from before.
- To solve all the problems is to have a map $p \to y$.

Synthetic programming

In synthetic programming, you have a type and you want a term of it.

- For example, you want a term of type $A \to (B \times C)$.
- The interface for a synthetic programmer can be given by $p : \mathbf{Poly}$.
- This is the "problems and solutions" semantics from before.
- To solve all the problems is to have a map $p \to y$.

The workflow is to farm out problems and aggregate solutions.

- A map $\varphi \colon p \to q_1 \otimes \cdots \otimes q_N$ is a two-step process:
  - Each problem $i \in p(1)$ is farmed out as $j_i \coloneqq \varphi_n(i) \in q_n(1)$.
  - Given a solution vector $(x_1, \ldots, x_N) \in (q_1 \otimes \cdots \otimes q_N)[j_1, \ldots, j_N]$
  - ...it returns a solution $\varphi_i^\sharp(x_1, \ldots, x_n) \in p[i]$.

Synthetic programming

In synthetic programming, you have a type and you want a term of it.

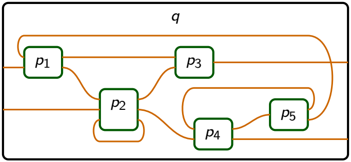
- For example, you want a term of type $A \to (B \times C)$.
- The interface for a synthetic programmer can be given by $p : \textbf{Poly}$.
- This is the "problems and solutions" semantics from before.
- To solve all the problems is to have a map $p \to \mathcal{y}$.

The workflow is to farm out problems and aggregate solutions.

- A map $\varphi \colon p \to q_1 \otimes \cdots \otimes q_N$ is a two-step process:
  - Each problem $i \in p(1)$ is farmed out as $j_i := \varphi_n(i) \in q_n(1)$.
  - Given a solution vector $(x_1, \ldots, x_N) \in (q_1 \otimes \cdots \otimes q_N)[j_1, \ldots, j_N]$
  - ...it returns a solution $\varphi_i^\sharp(x_1, \ldots, x_n) \in p[i]$.
- A map $\psi \colon p \to q_1 \triangleleft \cdots \triangleleft q_N$ is an $N + 1$-step process.
  - It sends each problem $i \in p(1)$ to a problem $j_1 \in q_1(1)$.
  - Given any solution $x_1 \in q_1[j_1]$, it returns a problem $j_2 \in q_2(1)$....
  - ...given any solution $x_N \in q_N[j_N]$, it returns a solution in $p[i]$.
- You get more examples by mixing and matching $\otimes, \triangleleft$.

The "scientist category"

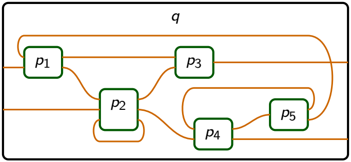
Now instead, imagine *walking through* a diagram like this:



- You're in some state as you walk into q through a door on the left.
- You're shuttled to some p-box, which you then enter.
    - You find the room in some state, you interact with it for a while.
    - You either never leave, or you leave out a door on the right.
    - When you leave, the room is in a new state, and so are you.
- You're shuttled either out the a q-door on the right or...
- ...to a new p-box, where the process repeats.

The "scientist category"

Now instead, imagine *walking through* a diagram like this:



- You're in some state as you walk into q through a door on the left.
- You're shuttled to some p-box, which you then enter.
  - You find the room in some state, you interact with it for a while.
  - You either never leave, or you leave out a door on the right.
  - When you leave, the room is in a new state, and so are you.
- You're shuttled either out the a q-door on the right or...
- ...to a new p-box, where the process repeats.

This is operadic, a different semantic than the simultaneous dyn. sys's.

- It has applications as a VDSL for rewriting protocols.
- It has a **Poly** description: the category $\mathbf{Set}_*$ is enriched in $\mathbf{Cat}^\sharp$.

Mathematical properties of Poly

Finally, the category **Poly** is incredibly rich and well-behaved:

Mathematical properties of Poly

Finally, the category **Poly** is incredibly rich and well-behaved:

- Coproducts and products that agree with usual polynomial arithmetic;
- All limits and colimits;
- At least three orthogonal factorization systems;
- A symmetric monoidal structure $\otimes$ distributing over $+$;
- A cartesian closure q^p and monoidal closure $[p, q]$ for $\otimes$;
- Another nonsymmetric monoidal structure $\triangleleft$ that's duoidal with $\otimes$;
- A left $\triangleleft$-coclosure $\left[\begin{smallmatrix} - \\ - \end{smallmatrix}\right]$, meaning $\mathbf{Poly}(p, q \triangleleft r) \cong \mathbf{Poly}(\left[\begin{smallmatrix} r \\ p \end{smallmatrix}\right], q)$;
- An indexed right $\triangleleft$-coclosure (Myers?), i.e. $\mathbf{Poly}(p, q \triangleleft r) \cong \sum_{f\colon\, p(1) \to q(1)} \mathbf{Poly}(p \overset{f}{\frown} q, r)$;
- An indexed right $\otimes$-coclosure (Niu?), i.e. $\mathbf{Poly}(p, q \otimes r) \cong \sum_{f\colon\, p(1) \to q(1)} \mathbf{Poly}(p \overset{f}{\nearrow} q, r)$;
- At least eight monoidal structures in total;
- $\triangleleft$-monoids generalize Σ-free operads;
- $\triangleleft$-comonoids are exactly categories; bicomodules are data migrations. This is $\mathbb{C}\mathbf{at}^\sharp$.

See "A reference for categ'ical structures on **Poly**", arXiv: 2202.00534

Summary

The **Poly** ecosystem incredibly rich. It has ready-made abstractions for:

- Moore machines, Mealy machines, wiring diagrams,
- Dependent type theory,
- Databases and data migration,
- Imperative programming (effects handling),
- Synthetic programming,
- Deep learning, etc.

Summary

The **Poly** ecosystem incredibly rich. It has ready-made abstractions for:

- Moore machines, Mealy machines, wiring diagrams,
- Dependent type theory,
- Databases and data migration,
- Imperative programming (effects handling),
- Synthetic programming,
- Deep learning, etc.

A very simple basic framework lends itself to a huge variety of applications.

- The same notation, techniques, theorems can be reused.
- And yet there are a lot of combinations.

Summary

The **Poly** ecosystem incredibly rich. It has ready-made abstractions for:

- Moore machines, Mealy machines, wiring diagrams,
- Dependent type theory,
- Databases and data migration,
- Imperative programming (effects handling),
- Synthetic programming,
- Deep learning, etc.

A very simple basic framework lends itself to a huge variety of applications.

- The same notation, techniques, theorems can be reused.
- And yet there are a lot of combinations.

It's a great abstraction, but how does it help with understanding health?

Outline

1. Introduction
2. Introduction to Poly
3. Unreasonable effectiveness of Poly
4. **What is health?**
    - Meeting the moment
    - Accountability
    - The health of this workshop: how to cohere?
5. Conclusion

Meeting the moment

It's unclear to me what health is or how to formally think about it.

- As said earlier, health means wholeness: a collective acts coherently.
- But then is a rock a degenerate case, or not of the right type?
- Should we distinguish between health, wealth, and wisdom?
- How are we going to find the right abstractions?

Meeting the moment

It's unclear to me what health is or how to formally think about it.

- As said earlier, health means wholeness: a collective acts coherently.
- But then is a rock a degenerate case, or not of the right type?
- Should we distinguish between health, wealth, and wisdom?
- How are we going to find the right abstractions?

We just spent a bunch of time with a certain mathematical abstraction.

- **Poly** can help us work with programming, data, dynamics, processes.
- These are useful in talking about various sorts of *system behavior*.
- So it'd be useful to think about how healthy systems *behave*.

Meeting the moment

It's unclear to me what health is or how to formally think about it.

- As said earlier, health means wholeness: a collective acts coherently.
- But then is a rock a degenerate case, or not of the right type?
- Should we distinguish between health, wealth, and wisdom?
- How are we going to find the right abstractions?

We just spent a bunch of time with a certain mathematical abstraction.

- **Poly** can help us work with programming, data, dynamics, processes.
- These are useful in talking about various sorts of *system behavior*.
- So it'd be useful to think about how healthy systems *behave*.

A healthy system is one that can *meet the moment*.

- When something threatens its cohesion, its integrity, it must meet it.
- It sees problems in advance, and it starts working to avoid them.

Accountability

What does it take for the collective to repeatedly see and solve problems?

- Different members of the collective need to work together.
- That means they need to communicate effectively.
- They need to be able to share their accounts of what's happening.
- If a member has failed, it needs to explain why it is still dependable.

Accountability

What does it take for the collective to repeatedly see and solve problems?

- Different members of the collective need to work together.
- That means they need to communicate effectively.
- They need to be able to share their accounts of what's happening.
- If a member has failed, it needs to explain why it is still dependable.

The collective's skill might be measured by its accounting ability.

- Religio = bind together. It's story-telling that orients and coheres.
- Math fields, as crystalized accounting systems, would be healthy.
- Cells developing protein signaling systems would be healthy.

Accountability

What does it take for the collective to repeatedly see and solve problems?

- Different members of the collective need to work together.
- That means they need to communicate effectively.
- They need to be able to share their accounts of what's happening.
- If a member has failed, it needs to explain why it is still dependable.

The collective's skill might be measured by its accounting ability.

- Religio = bind together. It's story-telling that orients and coheres.
- Math fields, as crystalized accounting systems, would be healthy.
- Cells developing protein signaling systems would be healthy.

Evolution has already produced many instances of accountability.

- (The word Evolution predates Darwin; it means *unfolding, unrolling*.)
- Genetic evolution made cells accountable to the multi-cell organism.
- Cultural evolution made people accountable to a moral or legal code.
- Lifetime evolution makes each person accountable to themselves.

Accountability

What does it take for the collective to repeatedly see and solve problems?

- Different members of the collective need to work together.
- That means they need to communicate effectively.
- They need to be able to share their accounts of what's happening.
- If a member has failed, it needs to explain why it is still dependable.

The collective's skill might be measured by its accounting ability.

- Religio = bind together. It's story-telling that orients and coheres.
- Math fields, as crystalized accounting systems, would be healthy.
- Cells developing protein signaling systems would be healthy.

Evolution has already produced many instances of accountability.

- (The word Evolution predates Darwin; it means *unfolding, unrolling*.)
- Genetic evolution made cells accountable to the multi-cell organism.
- Cultural evolution made people accountable to a moral or legal code.
- Lifetime evolution makes each person accountable to themselves.

In each case, language is developed to translate between part and whole.

The health of this workshop: how to cohere?

We are here to understand how collectives coordinate to meet the moment.

- We're in the midst of massive and unsettling change on Earth.
- Humanity needs to meet the moment.
- We need to coordinate, not disintegrate.
- We need a strongly accountable language so we can work together.

The health of this workshop: how to cohere?

We are here to understand how collectives coordinate to meet the moment.

- We're in the midst of massive and unsettling change on Earth.
- Humanity needs to meet the moment.
- We need to coordinate, not disintegrate.
- We need a strongly accountable language so we can work together.

As a collective ourselves, we also need to meet the moment.

- We want this workshop to be successful, to produce useful math.
- How do we work together to do that?
- By what method might we continually find and be compelled to use...
- ...clear, effective, well-oriented communication?

The health of this workshop: how to cohere?

We are here to understand how collectives coordinate to meet the moment.

- We're in the midst of massive and unsettling change on Earth.
- Humanity needs to meet the moment.
- We need to coordinate, not disintegrate.
- We need a strongly accountable language so we can work together.

As a collective ourselves, we also need to meet the moment.

- We want this workshop to be successful, to produce useful math.
- How do we work together to do that?
- By what method might we continually find and be compelled to use...
- ...clear, effective, well-oriented communication?

Throughout this workshop, we can use our own coherence as a guide.

- Notice a failure to cohere? Look for the right way to think about why.
- What's at the core of this whole inquiry? What do we need to clarify?

Outline

1. Introduction
2. Introduction to Poly
3. Unreasonable effectiveness of Poly
4. What is health?
5. **Conclusion**
   - Summary

Summary

Health is about wholeness: collectives that can work together.

- Language works in the sense of basic physics. “Pass the salt”.
- Our human collective works together by exchanging language.
- This language helps the collective *make sense* of local data.
- When a collective has a *sense*, it can work as a whole.

Summary

Health is about wholeness: collectives that can work together.

- Language works in the sense of basic physics. "Pass the salt".
- Our human collective works together by exchanging language.
- This language helps the collective *make sense* of local data.
- When a collective has a *sense*, it can work as a whole.

We need to find the right abstractions—the right language—for health.

- World problems can't be addressed unless recognized and understood.
- Healthy systems need to have good internal accounting.
- It's a meta-level problem to say what that means, FRA to discuss it.

Summary

Health is about wholeness: collectives that can work together.

- Language works in the sense of basic physics. “Pass the salt”.
- Our human collective works together by exchanging language.
- This language helps the collective *make sense* of local data.
- When a collective has a *sense*, it can work as a whole.

We need to find the right abstractions—the right language—for health.

- World problems can’t be addressed unless recognized and understood.
- Healthy systems need to have good internal accounting.
- It’s a meta-level problem to say what that means, FRA to discuss it.

I presented **Poly**, which is surprisingly abundant and applicable.

- Abundant: limits, colimits, monoidal structures, closures, coclosures...
- Applicable: dynamical systems, databases, synthetic programming.
- It may or may not be useful in thinking deeply about health.

Summary

Health is about wholeness: collectives that can work together.

- Language works in the sense of basic physics. “Pass the salt”.
- Our human collective works together by exchanging language.
- This language helps the collective *make sense* of local data.
- When a collective has a *sense*, it can work as a whole.

We need to find the right abstractions—the right language—for health.

- World problems can’t be addressed unless recognized and understood.
- Healthy systems need to have good internal accounting.
- It’s a meta-level problem to say what that means, FRA to discuss it.

I presented **Poly**, which is surprisingly abundant and applicable.

- Abundant: limits, colimits, monoidal structures, closures, coclosures...
- Applicable: dynamical systems, databases, synthetic programming.
- It may or may not be useful in thinking deeply about health.

I’m looking forward to FRA with you; may we meet the moment!

Thanks! Comments and questions welcome...