Poly: a category of remarkable abundance

David I. Spivak

TOPOS INSTITUTE

Colloquium
2021 February 04

Outline

My personal history with math

I've always believed I could understand self, life, and world with math.

- We generally share experience and knowledge in "natural language".
- Is any of it inherently precluded from mathematical expression?

My personal history with math

I've always believed I could understand self, life, and world with math.

- We generally share experience and knowledge in "natural language".
- Is any of it inherently precluded from mathematical expression?

When I learned CT, I thought "this is where I can say it all."

- It's a sublanguage of math that can talk about math.
- It's clean and principled and structural and expressive.

So I got to work trying to understand self, life, and world.

My personal history with ACT

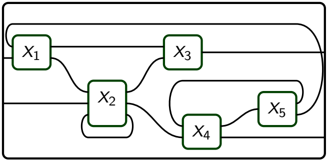
What can we say about self, life, and world?

- I first assumed everything is information and communication.
  - Pretend our minds are information-storage devices.
  - How do we communicate with each other and with reality?
  - Understand everything in terms of databases and data migration!
    - (Categories, set-valued functors, parametric right adjoints.)

My personal history with ACT

What can we say about self, life, and world?

- I first assumed everything is information and communication.
  - Pretend our minds are information-storage devices.
  - How do we communicate with each other and with reality?
  - Understand everything in terms of databases and data migration!
    - (Categories, set-valued functors, parametric right adjoints.)
  - But interacting processes didn't seem to fit nicely.
- So then I assumed everything is interacting dynamical systems.
  - It's machines sending each other information, all the way down.



  - But should they really be wired the same way forever?

My personal history with Poly

Then one day I met **Poly** and fell in love.

- It captures dynamical systems and "rewiring diagrams".
- As a category it's exceptionally well-behaved.

My personal history with Poly

Then one day I met **Poly** and fell in love.

- It captures dynamical systems and "rewiring diagrams".
- As a category it's exceptionally well-behaved.

The dynamics seemed to really be all about *comonoids* in **Poly**.

- Joachim Kock pointed me to R. Garner; I found his HoTTEST talk.
- Garner explained Ahman-Uustalu's result: "comonoids = categories"
- Garner also explained that bimodules = parametric right adjoints.

My personal history with Poly

Then one day I met **Poly** and fell in love.

- It captures dynamical systems and "rewiring diagrams".
- As a category it's exceptionally well-behaved.

The dynamics seemed to really be all about *comonoids* in **Poly**.

- Joachim Kock pointed me to R. Garner; I found his HoTTEST talk.
- Garner explained Ahman-Uustalu's result: "comonoids = categories"
- Garner also explained that bimodules = parametric right adjoints.

Suddenly everything I'd been working on for 13 years came together.

- I was overwhelmed by **Poly**'s elegance and capacity for application.
- It is extremely computational and hands-on...
- ...while displaying excellent formal properties.

Toward metaphysics

I use **Poly** to help ground my thinking about self, life, and world.

- What does it mean that I can "manipulate objects"?
- How should I think about biological reproduction?
- If it's always *now*, how do I perceive events that "unfold over time"?
- What is survival? If we change over time, what survives?

Toward metaphysics

I use **Poly** to help ground my thinking about self, life, and world.

- What does it mean that I can "manipulate objects"?
- How should I think about biological reproduction?
- If it's always *now*, how do I perceive events that "unfold over time"?
- What is survival? If we change over time, what survives?

I'm happy to talk with you about these ideas off-line.

Plan for the talk

Here's the plan for today's talk

- Theory
  - Define **Poly** and one of its monoidal structures
  - Comonoids = categories, coalgebras = copresheaves, etc
  - Monoids generalize operads, algebras = operad-algebras, etc

Plan for the talk

Here's the plan for today's talk

- Theory
  - Define **Poly** and one of its monoidal structures
  - Comonoids = categories, coalgebras = copresheaves, etc
  - Monoids generalize operads, algebras = operad-algebras, etc
- Applications
  - Dynamical systems
  - Databases
  - Cellular automata
- Conclusion
  - Upcoming events
  - Summary

Plan for the talk

Here's the plan for today's talk

- Theory
  - Define **Poly** and one of its monoidal structures
  - Comonoids = categories, coalgebras = copresheaves, etc
  - Monoids generalize operads, algebras = operad-algebras, etc
- Applications
  - Dynamical systems
  - Databases
  - Cellular automata
- Conclusion
  - Upcoming events
  - Summary

Think of the talk as a calling card: reach out if you want to discuss!

Outline

Poly for experts

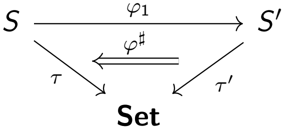
What I'll call the category **Poly** has many names.

- The free completely distributive category on one object;
- The free coproduct completion of $\mathbf{Set}^{\mathsf{op}}$;
- 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 free coproduct completion of $\mathbf{Set}^{\mathrm{op}}$;
- 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 category of *typed sets* and colax maps between them.
  - Objects: pairs (S, τ), where $S \in \mathbf{Set}$ and $\tau \colon S \to \mathbf{Set}$.
  - Morphisms $(S, \tau) \xrightarrow{\varphi} (S', \tau')$: pairs $(\varphi_1, \varphi^\sharp)$, where



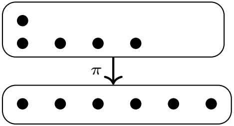
But let's make this easier.

What is a polynomial?

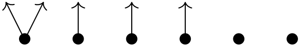
Algebraic

$y^2 + 3y + 2$

Bundle



Corolla forest

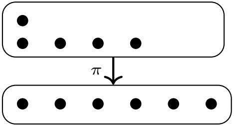


What is a polynomial?

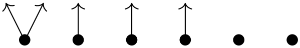
Algebraic

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

Bundle



Corolla forest



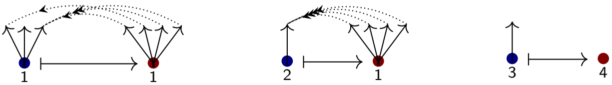
Interpretations:

- Each corolla in p is a decision; its leaves are the options.
- Each corolla in p is a position; its leaves are directions.

What is a morphism of polynomials?

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

A morphism $p \xrightarrow{\varphi} q$ delegates each p-decision to a q-decision, passing back options:



Example: how to think of a map $y^2 + y^6 \to y^{52}$.

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 $I \cong p(1)$. So we can write

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

Composition monoidal structure (**Poly**, $y, \triangleleft$)

The composite of two polynomial functors is again polynomial.

- Let's denote the composite of p and q by $p \triangleleft q$.
- Example: if $p := y^2$, $q := y + 1$, then $p \triangleleft q \cong y^2 + 2y + 1$.
- This is a monoidal structure, but not symmetric. $(q \triangleleft p \cong y^2 + 1)$
- The identity functor y is the unit: $p \triangleleft y \cong p \cong y \triangleleft p$.

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

The composite of two polynomial functors is again polynomial.

- Let's denote the composite of p and q by $p \triangleleft q$.
- Example: if $p := y^2$, $q := y + 1$, then $p \triangleleft q \cong y^2 + 2y + 1$.
- This is a monoidal structure, but not symmetric. $(q \triangleleft p \cong y^2 + 1)$
- The identity functor y is the unit: $p \triangleleft y \cong p \cong y \triangleleft p$.

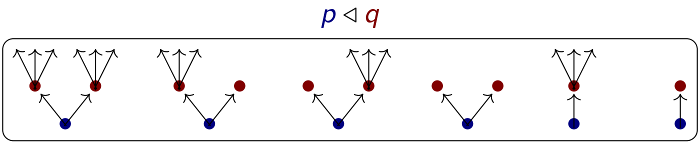
Why the we weird symbol $\triangleleft$ rather than $\circ$?

- We want to reserve $\circ$ for morphism composition.
- The notation $p \triangleleft q$ represents trees with p under q.

Composition given by stacking trees

Suppose $p \coloneqq y^2 + y$ and $q \coloneqq y^3 + 1$.

Draw the composite $p \triangleleft q$ by stacking q-trees on top of p-trees:



You can also read it as q feeding into p, which is how composition works.

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

In any monoidal category $(\mathcal{M}, I, \otimes)$, one can consider comonoids.

- A comonoid is a triple (m, ϵ, δ) satisfying certain rules, where
  - $m \in \mathcal{M}$ is an object, the *carrier*,
  - $\epsilon \colon m \to I$ is a map, the *counit*, and
  - $\delta \colon m \to m \otimes m$ is a map, the *comultiplication*.

In $(\mathbf{Poly}, y, \triangleleft)$, comonoids are exactly categories![1]

[1] Ahman-Uustalu. See my talk, https://www.youtube.com/watch?v=2mWnrgPIrlA

Comonoids in $(\textbf{Poly}, y, \triangleleft)$

In any monoidal category $(\mathcal{M}, I, \otimes)$, one can consider comonoids.

- A comonoid is a triple (m, ϵ, δ) satisfying certain rules, where
  - $m \in \mathcal{M}$ is an object, the *carrier*,
  - $\epsilon \colon m \to I$ is a map, the *counit*, and
  - $\delta \colon m \to m \otimes m$ is a map, the *comultiplication*.

In $(\textbf{Poly}, y, \triangleleft)$, comonoids are exactly categories![1]

- If $\mathcal{C}$ is a category, the corresponding comonoid is

$$\mathfrak{c} := \sum_{i \in \mathrm{Ob}(\mathcal{C})} y^{\mathfrak{c}[i]}$$

  where $\mathfrak{c}[i]$ is the set of morphisms in $\mathcal{C}$ that emanate from i.
- The counit $\epsilon \colon \mathfrak{c} \to y$ assigns to each object an identity.
- The comult $\delta \colon \mathfrak{c} \to \mathfrak{c} \triangleleft \mathfrak{c}$ assigns codomains and composites.

[1] Ahman-Uustalu. See my talk, https://www.youtube.com/watch?v=2mWnrgPIrlA

Comonoid maps are "cofunctors"

In **Poly**, comonoids are categories, but their morphisms aren't functors.

- A comonoid morphism $\varphi\colon \mathscr{C} \nrightarrow \mathscr{D}$ is called a *cofunctor*.
- It includes a **Poly** map on carriers. For each object $i \in \mathfrak{c}(1)$, we get:
  - an object $j \coloneqq \varphi_1(i) \in \mathfrak{d}(1)$ and
  - for each emanating $f \in \mathfrak{d}[j]$, an emanating $\varphi_i^\sharp(f) \in \mathfrak{c}[i]$.

Comonoid maps are "cofunctors"

In **Poly**, comonoids are categories, but their morphisms aren't functors.

- A comonoid morphism $\varphi\colon \mathcal{C} \nrightarrow \mathcal{D}$ is called a *cofunctor*.
- It includes a **Poly** map on carriers. For each object $i \in \mathfrak{c}(1)$, we get:
  - an object $j := \varphi_1(i) \in \mathfrak{d}(1)$ and
  - for each emanating $f \in \mathfrak{d}[j]$, an emanating $\varphi_i^\sharp(f) \in \mathfrak{c}[i]$.

Example: what is a cofunctor $\mathcal{C} \overset{\varphi}{\nrightarrow} y^{\mathbb{N}}$?

- It is trivial on objects. On morphisms...
- ...it assigns an emanating morphism $\varphi_i^\sharp(1)$ to each object $i \in \mathfrak{c}(1)$.

Comonoid maps are "cofunctors"

In **Poly**, comonoids are categories, but their morphisms aren't functors.

- A comonoid morphism $\varphi\colon \mathcal{C} \nrightarrow \mathcal{D}$ is called a *cofunctor*.
- It includes a **Poly** map on carriers. For each object $i \in \mathfrak{c}(1)$, we get:
  - an object $j := \varphi_1(i) \in \mathfrak{d}(1)$ and
  - for each emanating $f \in \mathfrak{d}[j]$, an emanating $\varphi_i^\sharp(f) \in \mathfrak{c}[i]$.

Example: what is a cofunctor $\mathcal{C} \overset{\varphi}{\nrightarrow} y^{\mathbb{N}}$?

- It is trivial on objects. On morphisms...
- ...it assigns an emanating morphism $\varphi_i^\sharp(1)$ to each object $i \in \mathfrak{c}(1)$.

"That's not what you do with a category!"

- Cofunctors are kinda weird right? A whole new world to explore.
- A cofunctor $\mathcal{C} \nrightarrow y^{\mathbb{N}}$ is like a vector field on the category.
- This hints at applications, which are coming soon.

Bicomodules in $(\mathbf{Poly}, y, \triangleleft)$

Given comonoids $\mathscr{C}, \mathscr{D}$, a $(\mathscr{C}, \mathscr{D})$-bicomodule is another kind of map.

- It's a polynomial m, equipped with two maps

$$\mathfrak{c} \triangleleft m \longleftarrow m \longrightarrow m \triangleleft \mathfrak{d}$$

  each cohering naturally with the comonoid structure ϵ, δ.
- I denote this $(\mathscr{C}, \mathscr{D})$-bicomodule m like so:

$$\mathfrak{c} \overset{m}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} \mathfrak{d} \qquad \text{or} \qquad \mathscr{C} \overset{m}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} \mathscr{D}$$

  - The $\triangleleft$'s at the ends help me remember the how the maps go.
  - Maybe it looks like it's going the wrong way, but hold on.

Bicomodules are parametric right adjoints

Garner explained[2] that bicomodules $m \in {}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}}$, which we've denoted

$$\mathcal{C} \xleftarrow{m} \mathcal{D}$$

can be identified with parametric right adjoint functors (prafunctors)

$$\mathcal{D}\text{-}\mathbf{Set} \xrightarrow{M} \mathcal{C}\text{-}\mathbf{Set}.$$

[2] Garner's HoTTEST video, https://www.youtube.com/watch?v=tW6HYnqn6eI

Bicomodules are parametric right adjoints

Garner explained[2] that bicomodules $m \in {}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}}$, which we've denoted

$$\mathcal{C} \overset{m}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} \mathcal{D}$$

can be identified with parametric right adjoint functors (prafunctors)

$$\mathcal{D}\text{-}\mathbf{Set} \xrightarrow{M} \mathcal{C}\text{-}\mathbf{Set}.$$

- From this perspective the arrow points in the expected direction.
- Check: ${}_{\mathcal{C}}\mathbf{Mod}_0 \cong \mathcal{C}\text{-}\mathbf{Set}$.

[2] Garner's HoTTEST video, `https://www.youtube.com/watch?v=tW6HYnqn6eI`

Bicomodules are parametric right adjoints

Garner explained[2] that bicomodules $m \in {}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}}$, which we've denoted

$$\mathcal{C} \overset{m}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} \mathcal{D}$$

can be identified with parametric right adjoint functors (prafunctors)

$$\mathcal{D}\text{-}\mathbf{Set} \xrightarrow{M} \mathcal{C}\text{-}\mathbf{Set}.$$

- From this perspective the arrow points in the expected direction.
- Check: ${}_{\mathcal{C}}\mathbf{Mod}_0 \cong \mathcal{C}\text{-}\mathbf{Set}$.

Prafunctors $\mathcal{C} \triangleleft\!\!\longrightarrow\!\!\triangleleft \mathcal{D}$ generalize profunctors $\mathcal{C} \nrightarrow \mathcal{D}$:

- A profunctor $\mathcal{C} \nrightarrow \mathcal{D}$ is a functor $\mathcal{C} \to (\mathcal{D}\text{-}\mathbf{Set})^{\mathsf{op}}$
- A prafunctor $\mathcal{C} \triangleleft\!\!\longrightarrow\!\!\triangleleft \mathcal{D}$ is a functor $\mathcal{C} \to \mathbf{Coco}\big((\mathcal{D}\text{-}\mathbf{Set})^{\mathsf{op}}\big)$...
- ...where **Coco** is the free coproduct completion.

I'll explain how to think about it concretely when we get to applications.

[2] Garner's HoTTEST video, `https://www.youtube.com/watch?v=tW6HYnqn6eI`

The framed bicategory $\mathbb{P}$

Poly comonoids, cofunctors, and bicomodules form a framed bicategory $\mathbb{P}$.

- It's got a ton of structure, e.g. two monoidal structures, $+, \otimes$.
- Despite the last slide, it's actually not that hard to think about.

Here are some facts about ${}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}}$ for categories $\mathcal{C}, \mathcal{D}$.

- ${}_{\mathcal{D}}\mathbf{Mod}_{\mathbf{0}} \cong \mathcal{D}\text{-}\mathbf{Set}$, copresheaves on $\mathcal{D}$.
- ${}_{\mathbf{1}}\mathbf{Mod}_{\mathcal{D}} \cong \mathbf{Coco}\big((\mathcal{D}\text{-}\mathbf{Set})^{\mathsf{op}}\big)$.
- ${}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}} \cong \mathbf{Cat}(\mathcal{C}, {}_{\mathbf{1}}\mathbf{Mod}_{\mathcal{D}})$.

The framed bicategory $\mathbb{P}$

Poly comonoids, cofunctors, and bicomodules form a framed bicategory $\mathbb{P}$.

- It's got a ton of structure, e.g. two monoidal structures, $+, \otimes$.
- Despite the last slide, it's actually not that hard to think about.

Here are some facts about ${}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}}$ for categories $\mathcal{C}, \mathcal{D}$.

- ${}_{\mathcal{D}}\mathbf{Mod}_{\mathbf{0}} \cong \mathcal{D}\text{-}\mathbf{Set}$, copresheaves on $\mathcal{D}$.
- ${}_{\mathbf{1}}\mathbf{Mod}_{\mathcal{D}} \cong \mathbf{Coco}\big((\mathcal{D}\text{-}\mathbf{Set})^{\mathsf{op}}\big)$.
- ${}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}} \cong \mathbf{Cat}(\mathcal{C}, {}_{\mathbf{1}}\mathbf{Mod}_{\mathcal{D}})$.

We can think about ${}_{\mathbf{1}}\mathbf{Mod}_{\mathcal{D}}$ as something like a polynomial rig in $\mathcal{D}$.

- If $\mathcal{D} = J$ is discrete, it's the rig of polynomials in variables $(y^j)_{j \in J}$.
- So ${}_I\mathbf{Mod}_J$ is I-many polynomials in J variables, as in Gambino-Kock.

The framed bicategory $\mathbb{P}$

Poly comonoids, cofunctors, and bicomodules form a framed bicategory $\mathbb{P}$.

- It's got a ton of structure, e.g. two monoidal structures, $+, \otimes$.
- Despite the last slide, it's actually not that hard to think about.

Here are some facts about ${}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}}$ for categories $\mathcal{C}, \mathcal{D}$.

- ${}_{\mathcal{D}}\mathbf{Mod}_{\mathbf{0}} \cong \mathcal{D}\text{-}\mathbf{Set}$, copresheaves on $\mathcal{D}$.
- ${}_{\mathbf{1}}\mathbf{Mod}_{\mathcal{D}} \cong \mathbf{Coco}\big((\mathcal{D}\text{-}\mathbf{Set})^{\mathsf{op}}\big)$.
- ${}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}} \cong \mathbf{Cat}(\mathcal{C}, {}_{\mathbf{1}}\mathbf{Mod}_{\mathcal{D}})$.

We can think about ${}_{\mathbf{1}}\mathbf{Mod}_{\mathcal{D}}$ as something like a polynomial rig in $\mathcal{D}$.

- If $\mathcal{D} = J$ is discrete, it's the rig of polynomials in variables $(\mathcal{y}^j)_{j \in J}$.
- So ${}_{I}\mathbf{Mod}_{J}$ is I-many polynomials in J variables, as in Gambino-Kock.
- For general $\mathcal{D}$, note that $\mathcal{y}^{-} \colon \mathcal{D} \to (\mathcal{D}\text{-}\mathbf{Set})^{\mathsf{op}}$ is free limit completion.
- So just generalize from sums of $\mathcal{D}$-products to sums of $\mathcal{D}$-limits, e.g.

$$\mathcal{y}^a \mathcal{y}^a + 42 \lim(\mathcal{y}^a \xrightarrow{f} \mathcal{y}^c \xleftarrow{g} \mathcal{y}^b) \quad \in {}_{\mathbf{1}}\mathbf{Mod}_{\mathcal{D}}$$

(Here, $f \colon a \to c$ and $g \colon b \to c$ are morphisms in $\mathcal{D}$).

Operads as monads in $\mathbb{P}$

In any framed bicategory, notation from $\mathbb{P}$, a monad $(\mathcal{C}, m, \eta, \mu)$ consists of

- An object $\mathcal{C}$, the *type*
- a bimodule $\mathcal{C} \overset{m}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} \mathcal{C}$, the *carrier*
- a 2-cell $\eta\colon \mathrm{id}_c \Rightarrow m$, the *unit*
- a 2-cell $\mu\colon m \circ m \Rightarrow m$, the *multiplication*
- satisfying the usual laws.

[3] Not quite the standard definition of operad, but one I like better: the input to a morphism is a set, rather than a list of objects. You can also talk about standard operads and generalizations within the $\mathbb{P}$ setting; see Gambino-Kock.

Operads as monads in $\mathbb{P}$

In any framed bicategory, notation from $\mathbb{P}$, a monad $(\mathcal{C}, m, \eta, \mu)$ consists of

- An object $\mathcal{C}$, the *type*
- a bimodule $\mathcal{C} \overset{m}{\triangleleft\!\!-\!\!\!-\!\!\triangleleft} \mathcal{C}$, the *carrier*
- a 2-cell $\eta\colon \mathrm{id}_c \Rightarrow m$, the *unit*
- a 2-cell $\mu\colon m \circ m \Rightarrow m$, the *multiplication*
- satisfying the usual laws.

In $\mathbb{P}$, these generalize operads in a number of ways:

- When $\mathcal{C} \cong I$ is discrete, $\eta^\sharp, \mu^\sharp$ are isos, you get colored operads.[3]
- Relaxing discreteness of $\mathcal{C}$, the input to a morphism can be...
- ... a diagram, rather than a mere set, of objects.
- Relaxing "iso" condition, composites and ids can have "weird" arities.

[3]Not quite the standard definition of operad, but one I like better: the input to a morphism is a set, rather than a list of objects. You can also talk about standard operads and generalizations within the $\mathbb{P}$ setting; see Gambino-Kock.

Grothendieck sites give $\mathbb{P}$-monads

Every Grothendieck site $(\mathcal{C}^{\mathsf{op}}, J)$ has an associated monad m_J in $\mathbb{P}$.

- A J-sheaf is an m_J-algebra, but not all m_J-algebras are J-sheaves.
- An m_J-algebra has existence, but not necess'ly uniqueness for gluing.

Grothendieck sites give $\mathbb{P}$-monads

Every Grothendieck site $(\mathcal{C}^{\text{op}}, J)$ has an associated monad m_J in $\mathbb{P}$.

- A J-sheaf is an m_J-algebra, but not all m_J-algebras are J-sheaves.
- An m_J-algebra has existence, but not necess'ly uniqueness for gluing.

To each Grothendieck top'y J, we need (m, η, μ) where $\mathcal{C} \triangleleft\!\xrightarrow{m}\!\triangleleft\ \mathcal{C}$.

- The topology J assigns to each $V \in \mathcal{C}$ a set J_V, "covering families"...
- ... and each $F \in J_V$ is assigned a subfunctor $S_F \subseteq \mathcal{C}[V]$.
- From this data we define $m \in$ **Poly**:

$$m := \sum_{V \in \text{Ob}(\mathcal{C})} \sum_{F \in J_V} y^{S_F}.$$

The Grothendieck top'y axioms endow the bimodule and monad structure.

Grothendieck sites give $\mathbb{P}$-monads

Every Grothendieck site $(\mathcal{C}^{\mathsf{op}}, J)$ has an associated monad m_J in $\mathbb{P}$.

- A J-sheaf is an m_J-algebra, but not all m_J-algebras are J-sheaves.
- An m_J-algebra has existence, but not necess'ly uniqueness for gluing.

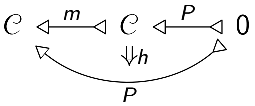
To each Grothendieck top'y J, we need (m, η, μ) where $\mathcal{C} \overset{m}{\triangleleft\!\!-\!\!\triangleleft} \mathcal{C}$.

- The topology J assigns to each $V \in \mathcal{C}$ a set J_V, "covering families"...
- ... and each $F \in J_V$ is assigned a subfunctor $S_F \subseteq \mathcal{C}[V]$.
- From this data we define $m \in$ **Poly**:

$$m := \sum_{V \in \mathrm{Ob}(\mathcal{C})} \sum_{F \in J_V} y^{S_F}.$$

The Grothendieck top'y axioms endow the bimodule and monad structure.

An algebra structure $m \circ P \xrightarrow{h} P$ assigns a section $h_V(F, s) \in P_V$ to each V-covering family F and matching family s of sections.



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, u, r, 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

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, u, r, 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.

This all works because $S\mathcal{y}^S$ is a comonoid.

Moore machines as maps in Poly

We can understand Moore machines 

in terms of polynomials.

- An uninitialized Moore machine $r\colon S \to B$ and $u\colon S \times A \to S$ is:
  - A map of polynomials $Sy^S \to By^A$.
  - φ_1 is the readout and $\varphi^\sharp$ is the update.
- Add initialization by giving a map $y \to Sy^S$.

Moore machines as maps in Poly

We can understand Moore machines 

in terms of polynomials.

- An uninitialized Moore machine $r\colon S \to B$ and $u\colon S \times A \to S$ is:
  - A map of polynomials $Sy^S \to By^A$.
  - φ_1 is the readout and $\varphi^\sharp$ is the update.
- Add initialization by giving a map $y \to Sy^S$.

A *p-dynamical system* allows different input-sets at different positions.

- For arbitrary $p \in \textbf{Poly}$ we can interpret a map $\varphi\colon Sy^S \to p$ as:
  - a readout: every state $s \in S$ gets a position $i := \varphi_1(s) \in p(1)$
  - an update: for every direction $d \in p[i]$, a next state $\varphi^\sharp_s(d) \in S$.
- Again, add initialization by giving a map $y \to Sy^S$.

Moore machines as maps in Poly

We can understand Moore machines 

in terms of polynomials.

- An uninitialized Moore machine $r\colon S \to B$ and $u\colon S \times A \to S$ is:
  - A map of polynomials $Sy^S \to By^A$.
  - φ_1 is the readout and $\varphi^\sharp$ is the update.
- Add initialization by giving a map $y \to Sy^S$.

A *p-dynamical system* allows different input-sets at different positions.

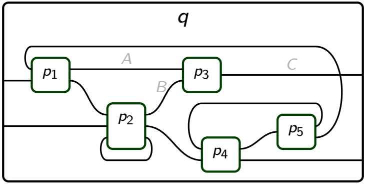
- For arbitrary $p \in \textbf{Poly}$ we can interpret a map $\varphi\colon Sy^S \to p$ as:
  - a readout: every state $s \in S$ gets a position $i := \varphi_1(s) \in p(1)$
  - an update: for every direction $d \in p[i]$, a next state $\varphi^\sharp_s(d) \in S$.
- Again, add initialization by giving a map $y \to Sy^S$.

Even more general: $Sy^S \nrightarrow \mathcal{C}$ for any category $\mathcal{C}$.

- For example, a map $Sy^S \to p$ can be identified with a cofunctor...
- ... $Sy^S \nrightarrow \textbf{Cofree}_p$,, where $\textbf{Cofree}_p$ is the *cofree comonoid* on p.

Wiring diagrams

We can have a bunch of dynamical systems interacting in an open system.



(φ)

Each box represents a monomial, e.g. $p_3 = Cy^{AB} \in \mathbf{Poly}$.

- The whole interaction, p_1 sending outputs to p_2 and p_3, etc....
- ... is captured by a map of polynomials $\varphi\colon p_1 \otimes \cdots \otimes p_5 \to q$. [4]
  - Given the positions (outputs) of each p_i, we get an output of q...
  - ... and when given an input of q, each p_i gets an input.

[4] Here $p \otimes p'$ just multiplies positions and directions,

$$p \otimes p' = \sum_{(i,i') \in p(1) \times p'(1)} y^{p[i] \times p'[i']}.$$

More general interaction

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 for $W \in \mathbf{Set}$.
- 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 $w \in W$ it receives.

Other sorts of dynamical systems

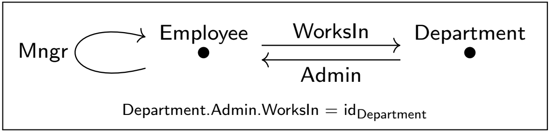
Dynamical systems are usually defined as actions of a monoid T.

- Discrete: $\mathbb{N}$, reversible: $\mathbb{Z}$, real-time: $\mathbb{R}$.
- If T is a monoid and S is a set, a T-action on S is equivalently...
- ... a map $S \times T \to S$ satisfying two laws, which is equivalently...
- ... a cofunctor $Sy^S \nrightarrow y^T$, as in our general definition above.

Categorical databases

One view on databases is that they're basically just copresheaves.

$\mathcal{C} :=$



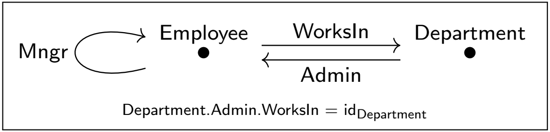
A functor $I\colon \mathcal{C} \to \mathbf{Set}$ (i.e. $\mathcal{C} \triangleleft\!\xrightarrow{I}\!\triangleleft\ 0$) can be represented as follows:

Employee	**WorksIn**	**Mngr**
♡	P9	♡
T****	bLue	orca
orca	bLue	orca

Department	**Admin**
bLue	T****
P9	♡

Categorical databases

One view on databases is that they're basically just copresheaves.

$\mathcal{C} :=$ 

A functor $I\colon \mathcal{C} \to \mathbf{Set}$ (i.e. $\mathcal{C} \overset{I}{\lhd\!\!\longrightarrow\!\!\lhd} 0$) can be represented as follows:

Employee	**WorksIn**	**Mngr**
♡	P9	♡
T****	bLue	orca
orca	bLue	orca

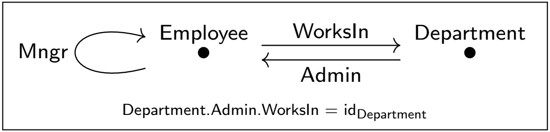
Department	**Admin**
bLue	T****
P9	♡

But where's the data? What are the employees names, etc.?

Categorical databases

One view on databases is that they're basically just copresheaves.

$\mathcal{C} :=$



More realistically, data should include *attributes* and look like this:

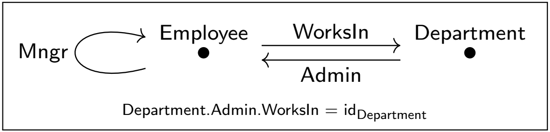
Employee	**FName**	**WorksIn**	**Mngr**
1	Alan	101	2
2	Ruth	101	2
3	Sara	102	3

Department	**DName**	**Secr**
101	Sales	1
102	IT	3

Categorical databases

One view on databases is that they're basically just copresheaves.

$\mathcal{C} :=$

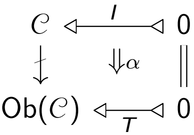


More realistically, data should include *attributes* and look like this:

Employee	FName	WorksIn	Mngr
1	Alan	101	2
2	Ruth	101	2
3	Sara	102	3

Department	DName	Secr
101	Sales	1
102	IT	3

- Assign a copresheaf $T \colon \mathrm{Ob}(\mathcal{C}) \to \mathbf{Set}$, e.g. $T(\text{Employee}) = \text{String}$.
- Using the canonical cofunctor $\mathcal{C} \nrightarrow \mathrm{Ob}(\mathcal{C})$, attributes are given by α:



Data migration

The framed bicategory structure of $\mathbb{P}$ is very useful in databases.

- We hinted at this in the last slide, adding attributes via a cofunctor.
- But so-called *data migration functors* are precisely prafunctors.

Data migration

The framed bicategory structure of $\mathbb{P}$ is very useful in databases.

- We hinted at this in the last slide, adding attributes via a cofunctor.
- But so-called *data migration functors* are precisely prafunctors.

A prafunctor $\mathcal{C} \xleftarrow{P} \mathcal{D}$ in ${}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}}$ can be understood as follows.

- First, it's a functor $\mathcal{C} \to {}_{\mathbf{1}}\mathbf{Mod}_{\mathcal{D}}$, so what's that?
- We said it's a formal coproduct of formal limits in $\mathcal{D}$.
- A formal limit in $\mathcal{D}$ is called a *conjunctive query* on $\mathcal{D}$.
- So a prafunctor $\mathbf{1} \xleftarrow{Q} \mathcal{D}$ is a disjoint union of conjunctive queries.
- Let's call Q a duc-query on $\mathcal{D}$.

Data migration

The framed bicategory structure of $\mathbb{P}$ is very useful in databases.

- We hinted at this in the last slide, adding attributes via a cofunctor.
- But so-called *data migration functors* are precisely prafunctors.

A prafunctor $\mathcal{C} \xleftarrow{P} \mathcal{D}$ in ${}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}}$ can be understood as follows.

- First, it's a functor $\mathcal{C} \to {}_{\mathbf{1}}\mathbf{Mod}_{\mathcal{D}}$, so what's that?
- We said it's a formal coproduct of formal limits in $\mathcal{D}$.
- A formal limit in $\mathcal{D}$ is called a *conjunctive query* on $\mathcal{D}$.
- So a prafunctor $\mathbf{1} \xleftarrow{Q} \mathcal{D}$ is a disjoint union of conjunctive queries.
- Let's call Q a duc-query on $\mathcal{D}$.

Example: if $\mathcal{D} = \left(\overset{\text{City}}{\bullet} \xrightarrow{\text{in}} \overset{\text{State}}{\bullet} \xleftarrow{\text{in}} \overset{\text{County}}{\bullet} \right)$, a duc-query might be...

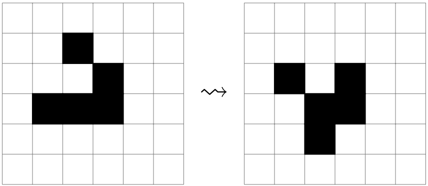
$$(\text{City} \times_{\text{State}} \text{City}) + (\text{City} \times_{\text{State}} \text{County}) + (\text{County} \times_{\text{State}} \text{County})$$

A general bimodule $P \in {}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}}$ is a $\mathcal{C}$-indexed duc-query on $\mathcal{D}$.

Cellular automata

The last thing we'll discuss today is cellular automata.

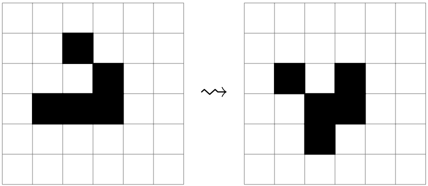
- Here's a picture of a *glider* from Conway's Game of Life:



Cellular automata

The last thing we'll discuss today is cellular automata.

- Here's a picture of a *glider* from Conway's Game of Life:

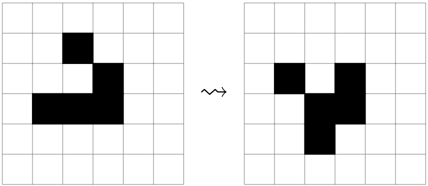


- GoL takes place on a grid, a set $V := \mathbb{Z} \times \mathbb{Z}$ of "squares"
- Each square has neighbors; think of the grid as a graph $A \rightrightarrows V$.
- Each square can be in one of two states: white or black.

Cellular automata

The last thing we'll discuss today is cellular automata.

- Here's a picture of a *glider* from Conway's Game of Life:



- GoL takes place on a grid, a set $V := \mathbb{Z} \times \mathbb{Z}$ of "squares"
- Each square has neighbors; think of the grid as a graph $A \rightrightarrows V$.
- Each square can be in one of two states: white or black.
- The state at any square is updated according to a formula, e.g.

  If the square is ■ *and has 2 or 3* ■ *neighbors, it stays* ■.
  If the square is □ *and has 3* ■ *neighbors, it turns* ■.
  Otherwise it turns / remains □.

Cellular automata as algebras in $\mathbb{P}$

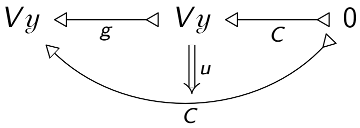
How do we encode this in $\mathbb{P}$?

- We encode the graph $A \rightrightarrows V$ as a prafunctor $Vy \xleftarrow{g} Vy$
  - Each $v \in V$ queries its neighbors (and itself).
  - The carrier of the prafunctor for GoL is $g := Vy^9$.

Cellular automata as algebras in $\mathbb{P}$

How do we encode this in $\mathbb{P}$?

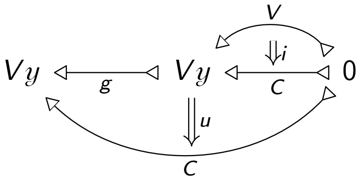
- We encode the graph $A \rightrightarrows V$ as a prafunctor $Vy \xleftarrow{g} Vy$
  - Each $v \in V$ queries its neighbors (and itself).
  - The carrier of the prafunctor for GoL is $g := Vy^9$.
- We encode the color-set for each node as a prafunctor $Vy \xleftarrow{C} 0$
  - In GoL, each $v \in V$ gets the set 2; i.e. $C := 2V$.
- We encode the update formula as a map u of prafunctors



Cellular automata as algebras in $\mathbb{P}$

How do we encode this in $\mathbb{P}$?

- We encode the graph $A \rightrightarrows V$ as a prafunctor $Vy \xleftarrow{g} Vy$
  - Each $v \in V$ queries its neighbors (and itself).
  - The carrier of the prafunctor for GoL is $g := Vy^9$.
- We encode the color-set for each node as a prafunctor $Vy \xleftarrow{C} 0$
  - In GoL, each $v \in V$ gets the set 2; i.e. $C := 2V$.
- We encode the update formula as a map u of prafunctors
- And we encode the initial color setup as a point $V \to C$:



From here you can iteratively "run" the cellular automaton.

Outline

Future work

Our society is engaging in many unhealthy behaviors.

- We need to understand what healthy behavior is.
- What activities are necessary for survival?
- How should we communicate so that what needs to happen does?

Future work

Our society is engaging in many unhealthy behaviors.

- We need to understand what healthy behavior is.
- What activities are necessary for survival?
- How should we communicate so that what needs to happen does?

We can think about them mathematically and apply our results.

- If we make philosophical/mathematical progress, we can echo it in tech
- If we make technological progress, people will take it up.
- If people use healthier tech systems, it might help.

Future work

Our society is engaging in many unhealthy behaviors.

- We need to understand what healthy behavior is.
- What activities are necessary for survival?
- How should we communicate so that what needs to happen does?

We can think about them mathematically and apply our results.

- If we make philosophical/mathematical progress, we can echo it in tech
- If we make technological progress, people will take it up.
- If people use healthier tech systems, it might help.

It is as promising a direction as anything I know of.

Workshop on polynomial functors in March

Joachim Kock and I are organizing a **Poly** workshop.[5]

- Dates: March 15 – 19
- Speakers:

Thorsten Altenkirch	Steve Awodey
Michael Batanin	Bryce Clarke
Marcelo Fiore	Richard Garner
David Gepner	Helle Hvid Hansen
Rune Haugseng	Bart Jacobs
André Joyal	Fredrik Nordvall-Forsberg
Kristina Sojakova	David Spivak
Ross Street	Tarmo Uustalu

[5] https://topos.site/p-func-2021-workshop/

Future Topos Institute colloquia

This is the first of a series of Topos Institute colloquia.

- More info here: `https://topos.site/seminars/`
- Next few speakers
  - Richard Garner
  - Gunnar Carlsson
  - Samson Abramsky

Please join us!

Summary

Poly is a category of remarkable abundance.

- It's completely combinatorial.
  - Calculations are concrete.
  - Much is already familiar, e.g. $(y+1)^2 \cong y^2 + 2y + 1$.
- It's theoretically beautiful.
  - Comonoids are categories, coalgebras are copresheaves.
  - Monoids generalize operads.
- It's got a wide scope of applicatons.
  - Databases and data migration.
  - Dynamical systems and cellular automata.

A single setting for pursuing real philosophical and technological progress.

Thanks! Questions and comments welcome.