

The Polynomial Abacus

David I. Spivak



Workshop on Polynomial Functors
2021 March 15 – 19

Outline

1 Introduction

- The abacus
- Plan

2 Theory

3 Applications

4 Conclusion

Abacus for the Glass Bead Game

There is a story by Herman Hesse, called *The Glass Bead Game*.

- It depicts a monastic community of thinkers, led by a “game master”.
- The game is played on an instrument involving strings of glass beads.

Like a rap battle or poetry slam, the game is played to express deep ideas.

- Players represent connections between math, music, philosophy, etc.
- The moving glass beads weave these subjects together in harmony.
- To play well is to contemplate and communicate profound insights.

Abacus for the Glass Bead Game

There is a story by Herman Hesse, called *The Glass Bead Game*.

- It depicts a monastic community of thinkers, led by a “game master”.
- The game is played on an instrument involving strings of glass beads.

Like a rap battle or poetry slam, the game is played to express deep ideas.

- Players represent connections between math, music, philosophy, etc.
- The moving glass beads weave these subjects together in harmony.
- To play well is to contemplate and communicate profound insights.

I loved the idea of the book, but something was missing.

- Hesse only roughly describes the instrument—the abacus—itself.
- What sort of combinatorial object is capable of this grand scope?

To my lights, **Poly** can serve as an abacus; I hope to justify that to you.

Approximate plan for tutorial

Today:

- Introduce **Poly** and its combinatorics (how the abacus works);
- Discuss its pleasing properties and monoidal structures;
- Present the framed bicategory \mathbb{P} .

Wednesday:

- Recall \mathbb{P} and discuss some properties of it;
- Consider applications: dynamical systems, data, and deep learning;
- Conclude with a summary.

Outline

1 Introduction

2 Theory

- **Poly** as a category
- A quick tour of **Poly**
- Comonoids in **Poly**
- The framed bicategory \mathbb{P}
- Monads in \mathbb{P}

3 Applications

4 Conclusion

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 **Set**^{op};
- The full subcategory of **[Set, Set]** spanned by functors that preserve connected limits;
- The full subcategory of **[Set, 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 **Set**^{op};
- The full subcategory of [**Set**, **Set**] spanned by functors that preserve connected limits;
- The full subcategory of [**Set**, **Set**] spanned by coproducts of repr'bles;
- The category of *typed sets* and colax maps between them.
 - Objects: *pairs* (I, τ) , where $I \in \mathbf{Set}$ and $\tau: I \rightarrow \mathbf{Set}$.
 - Morphisms $(I, \tau) \xrightarrow{\varphi} (I', \tau')$: *pairs* $(\varphi_1, \varphi^\#)$, where

$$\begin{array}{ccc}
 I & \xrightarrow{\varphi_1} & I' \\
 \searrow \tau & \xleftarrow{\varphi^\#} & \swarrow \tau' \\
 & \mathbf{Set} &
 \end{array}$$

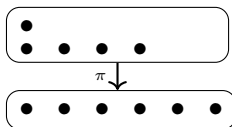
But let's make this easier.

What is a polynomial?

Algebraic

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

Bundle



Corolla forest

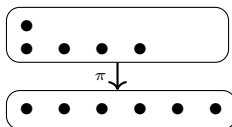


What is a polynomial?

Algebraic

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

Bundle



Corolla forest



One could repurpose this machine to represent $15y^{5 \times 2} \in \mathbf{Poly}$.

Terminology woes

Issue: prior terminology from computer science doesn't fit my conception.

$$p := y^3 + y^2 + y^2 + 1$$

- Container terminology from Abbott: “**shapes** and **positions**”.
 - `data p Y = Foo Y Y Y | Bar Y Y | Baz Y Y | Qux`
 - Container p has four “**shapes**”, e.g. Foo has three “**positions**”.

Terminology woes

Issue: prior terminology from computer science doesn't fit my conception.

$$p := y^3 + y^2 + y^2 + 1$$

- Container terminology from Abbott: “**shapes** and **positions**”.
- `data p Y = Foo Y Y Y | Bar Y Y | Baz Y Y | Qux`
- Container p has four “**shapes**”, e.g. `Foo` has three “**positions**”.
- We prefer to think of these “**positions**” as *projection arrows*.



Terminology woes

Issue: prior terminology from computer science doesn't fit my conception.

$$p := y^3 + y^2 + y^2 + 1$$

- Container terminology from Abbott: “**shapes** and **positions**”.
 - `data p Y = Foo Y Y Y | Bar Y Y | Baz Y Y | Qux`
 - Container p has four “**shapes**”, e.g. `Foo` has three “**positions**”.
 - We prefer to think of these “**positions**” as *projection arrows*.



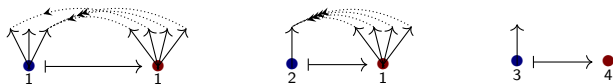
- Hard decision but I'll say **positions** and **directions**. Reasons:
 - Dynamical systems: **position** = point, **direction** = vector.
 - Categories: **position** = object, **direction** = morphism.
 - Terminal coalgebra trees: **position** = label, **direction** = arrow.

Combinatorics of polynomial morphisms

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



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



Example: how to think of

- $y^2 + y^6 \rightarrow y^{52}$?
- $p \rightarrow y$ for arbitrary p ?

The category of polynomials

Easiest description: **Poly** = “sums of representables functors **Set** \rightarrow **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.

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

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

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

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

Here's a derivation of the combinatorial formula for morphisms:

$$\begin{aligned} \mathbf{Poly}(p, q) &= \mathbf{Poly} \left(\sum_{i \in p(1)} y^{p[i]}, \sum_{j \in q(1)} y^{q[j]} \right) \cong \prod_{i \in p(1)} \mathbf{Poly} \left(y^{p[i]}, \sum_{j \in q(1)} y^{q[j]} \right) \\ &\cong \prod_{i \in p(1)} \sum_{j \in q(1)} \mathbf{Set}(q[j], p[i]) \end{aligned}$$

"For each $i \in p(1)$, a choice of $j \in q(1)$ and a function $q[j] \rightarrow p[i]$."

Notation for the abacus

For any polynomial $p \in \mathbf{Poly}$, I'll use the following sort of notation

$$\begin{array}{c} p[-] \\ p(1) \end{array} \begin{array}{|c|} \hline \\ \hline \\ \hline \end{array}$$

Notation for the abacus

For any polynomial $p \in \mathbf{Poly}$, I'll use the following sort of notation

$$\begin{array}{c} p[-] \\ p(1) \end{array} \begin{array}{|c|} \hline \\ \hline i \\ \hline \end{array}$$

- The bottom part is filled by indicating a position, say $i \in p(1)$.

Notation for the abacus

For any polynomial $p \in \mathbf{Poly}$, I'll use the following sort of notation

$$\begin{array}{c|c} p[-] & d \\ \hline p(1) & i \end{array}$$

- The bottom part is filled by indicating a position, say $i \in p(1)$.
- *Only then* can the top part be filled by a direction, say $d \in p[i]$.

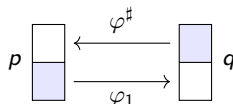
Notation for the abacus

For any polynomial $p \in \mathbf{Poly}$, I'll use the following sort of notation

$$\begin{array}{c} p[-] \\ p(1) \end{array} \begin{array}{|c|} \hline d \\ \hline i \\ \hline \end{array}$$

- The bottom part is filled by indicating a position, say $i \in p(1)$.
- *Only then* can the top part be filled by a direction, say $d \in p[i]$.

This gets more interesting for maps. A map $\varphi: p \rightarrow q$ is shown:



The map φ is a formula saying “however you fill blue’s, I’ll fill whites.”

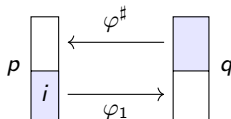
Notation for the abacus

For any polynomial $p \in \mathbf{Poly}$, I'll use the following sort of notation

$$\begin{array}{c} p[-] \\ p(1) \end{array} \begin{array}{|c|} \hline d \\ \hline i \\ \hline \end{array}$$

- The bottom part is filled by indicating a position, say $i \in p(1)$.
- *Only then* can the top part be filled by a direction, say $d \in p[i]$.

This gets more interesting for maps. A map $\varphi: p \rightarrow q$ is shown:



The map φ is a formula saying “however you fill blue’s, I’ll fill whites.”

- For any $i \in p(1)$ you choose,

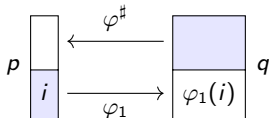
Notation for the abacus

For any polynomial $p \in \mathbf{Poly}$, I'll use the following sort of notation

$$\begin{array}{c} p[-] \\ p(1) \end{array} \begin{array}{|c|} \hline d \\ \hline i \\ \hline \end{array}$$

- The bottom part is filled by indicating a position, say $i \in p(1)$.
- *Only then* can the top part be filled by a direction, say $d \in p[i]$.

This gets more interesting for maps. A map $\varphi: p \rightarrow q$ is shown:



The map φ is a formula saying “however you fill blue’s, I’ll fill whites.”

- For any $i \in p(1)$ you choose, I’ll return $\varphi_1(i) \in q(1)$, and

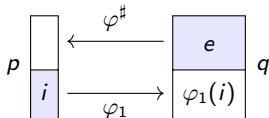
Notation for the abacus

For any polynomial $p \in \mathbf{Poly}$, I'll use the following sort of notation

$$\begin{array}{c} p[-] \\ p(1) \end{array} \begin{array}{|c|} \hline d \\ \hline i \\ \hline \end{array}$$

- The bottom part is filled by indicating a position, say $i \in p(1)$.
- *Only then* can the top part be filled by a direction, say $d \in p[i]$.

This gets more interesting for maps. A map $\varphi: p \rightarrow q$ is shown:



The map φ is a formula saying “however you fill blue’s, I’ll fill whites.”

- For any $i \in p(1)$ you choose, I’ll return $\varphi_1(i) \in q(1)$, and
- for any $e \in q[\varphi_1(i)]$ you choose,

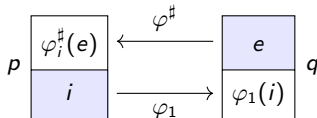
Notation for the abacus

For any polynomial $p \in \mathbf{Poly}$, I'll use the following sort of notation

$$\begin{array}{c} p[-] \\ p(1) \end{array} \begin{array}{|c|} \hline d \\ \hline i \\ \hline \end{array}$$

- The bottom part is filled by indicating a position, say $i \in p(1)$.
- *Only then* can the top part be filled by a direction, say $d \in p[i]$.

This gets more interesting for maps. A map $\varphi: p \rightarrow q$ is shown:



The map φ is a formula saying “however you fill blue’s, I’ll fill whites.”

- For any $i \in p(1)$ you choose, I’ll return $\varphi_1(i) \in q(1)$, and
- for any $e \in q[\varphi_1(i)]$ you choose, I’ll return $\varphi_i^\#(e) \in p[i]$.

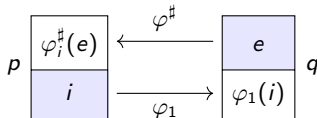
Notation for the abacus

For any polynomial $p \in \mathbf{Poly}$, I'll use the following sort of notation

$$\begin{array}{c} p[-] \\ p(1) \end{array} \begin{array}{c} d \\ i \end{array}$$

- The bottom part is filled by indicating a position, say $i \in p(1)$.
- *Only then* can the top part be filled by a direction, say $d \in p[i]$.

This gets more interesting for maps. A map $\varphi: p \rightarrow q$ is shown:



The map φ is a formula saying “however you fill blue’s, I’ll fill whites.”

- For any $i \in p(1)$ you choose, I’ll return $\varphi_1(i) \in q(1)$, and
- for any $e \in q[\varphi_1(i)]$ you choose, I’ll return $\varphi_i^\#(e) \in p[i]$.

But this notation will really come in handy later in handling composition.

Pleasing aspects of Poly

Here are some properties enjoyed by **Poly**:

- **Poly** contains two copies of **Set** and one copy of **Set**^{op}.
 - Sets A can be represented as a constant or linear: $A, Ay \in \mathbf{Poly}$.
 - Sets A can be op-represented as representables $y^A \in \mathbf{Poly}$.
 - Each of these inclusions is full and has at least one adjoint.

Pleasing aspects of Poly

Here are some properties enjoyed by **Poly**:

- **Poly** contains two copies of **Set** and one copy of **Set**^{op}.
 - Sets A can be represented as a constant or linear: $A, Ay \in \mathbf{Poly}$.
 - Sets A can be op-represented as representables $y^A \in \mathbf{Poly}$.
 - Each of these inclusions is full and has at least one adjoint.
- **Poly** has all coproducts and limits (extensive), and is Cartesian closed;
 - These agree with coproducts, limits, closure in “**Set**^{Set}”.
 - 0 is initial, 1 is terminal, $+$ is coproduct, \times is product.
 - y^A is internal hom between $A, y \in \mathbf{Poly}$. For fun: $y^y \cong y + 1$.
- **Poly** has coequalizers, though these differ from coeq's in “**Set**^{Set}”.

Pleasing aspects of Poly

Here are some properties enjoyed by **Poly**:

- **Poly** contains two copies of **Set** and one copy of **Set**^{op}.
 - Sets A can be represented as a constant or linear: $A, Ay \in \mathbf{Poly}$.
 - Sets A can be op-represented as representables $y^A \in \mathbf{Poly}$.
 - Each of these inclusions is full and has at least one adjoint.
- **Poly** has all coproducts and limits (extensive), and is Cartesian closed;
 - These agree with coproducts, limits, closure in “**Set**^{Set}”.
 - 0 is initial, 1 is terminal, $+$ is coproduct, \times is product.
 - y^A is internal hom between $A, y \in \mathbf{Poly}$. For fun: $y^y \cong y + 1$.
- **Poly** has coequalizers, though these differ from coeq's in “**Set**^{Set}”.
- **Poly** has two factorization systems: epi-mono, vertical-cartesian.

Monoidal structures on Poly

There are many monoidal structures on **Poly**.

- It has a coproduct $(0, +)$ structure.
- Day convolution can be applied to any SMC structure (I, \cdot) on **Set**.
 - The result is a distributive monoidal structure (y^I, \odot) on **Poly**.
 - In the case of $(0, +)$, the result is the product $(1, \times)$.
 - In the case of $(1, \times)$, the result is (y, \otimes) .

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

Monoidal structures on Poly

There are many monoidal structures on **Poly**.

- It has a coproduct $(0, +)$ structure.
- Day convolution can be applied to any SMC structure (I, \cdot) on **Set**.
 - The result is a distributive monoidal structure (y^I, \odot) on **Poly**.
 - In the case of $(0, +)$, the result is the product $(1, \times)$.
 - In the case of $(1, \times)$, the result is (y, \otimes) .

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

- The \otimes product has a closure (internal hom) $[-, -]$ given by

$$[p, q] := \sum_{\varphi: p \rightarrow q} y^{\sum_{i \in p(1)} q[\varphi_1(i)]}$$

Monoidal structures on Poly

There are many monoidal structures on **Poly**.

- It has a coproduct $(0, +)$ structure.
- Day convolution can be applied to any SMC structure (I, \cdot) on **Set**.
 - The result is a distributive monoidal structure (y^I, \odot) on **Poly**.
 - In the case of $(0, +)$, the result is the product $(1, \times)$.
 - In the case of $(1, \times)$, the result is (y, \otimes) .

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

- The \otimes product has a closure (internal hom) $[-, -]$ given by

$$[p, q] := \sum_{\varphi: p \rightarrow q} y^{\sum_{i \in p(1)} q[\varphi_1(i)]}$$

There's one more monoidal product, which will be of great interest.

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

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 $(\text{Poly}, y, \triangleleft)$

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 ?

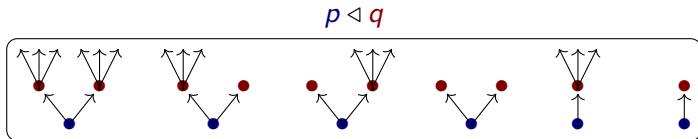
- We want to reserve \circ for morphism composition.
- The notation $p \triangleleft q$ represents trees with p under q .

Composition given by stacking trees

Suppose $p := y^2 + y$ and $q := y^3 + 1$.



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.

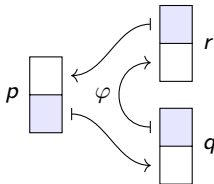
Maps to composites

The abacus pictures are most useful for maps $p \rightarrow q_1 \triangleleft \cdots \triangleleft q_k$.

- A map $\varphi: p \rightarrow q \triangleleft r$ is an element of

$$\varphi \in \mathbf{Poly}(p, q \triangleleft r) \cong \prod_{i \in p(1)} \sum_{j \in q(1)} \prod_{e \in q[j]} \sum_{k \in r(1)} \prod_{f \in r[k]} \sum_{d \in p[i]} 1.$$

We could write it with our abacus pictures:



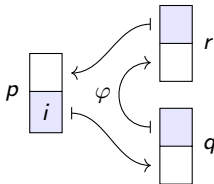
Maps to composites

The abacus pictures are most useful for maps $p \rightarrow q_1 \triangleleft \cdots \triangleleft q_k$.

- A map $\varphi: p \rightarrow q \triangleleft r$ is an element of

$$\varphi \in \mathbf{Poly}(p, q \triangleleft r) \cong \prod_{i \in p(1)} \sum_{j \in q(1)} \prod_{e \in q[j]} \sum_{k \in r(1)} \prod_{f \in r[k]} \sum_{d \in p[i]} 1.$$

We could write it with our abacus pictures:



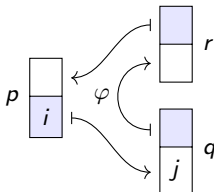
Maps to composites

The abacus pictures are most useful for maps $p \rightarrow q_1 \triangleleft \cdots \triangleleft q_k$.

- A map $\varphi: p \rightarrow q \triangleleft r$ is an element of

$$\varphi \in \mathbf{Poly}(p, q \triangleleft r) \cong \prod_{i \in p(1)} \sum_{j \in q(1)} \prod_{e \in q[j]} \sum_{k \in r(1)} \prod_{f \in r[k]} \sum_{d \in p[i]} 1.$$

We could write it with our abacus pictures:



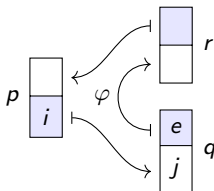
Maps to composites

The abacus pictures are most useful for maps $p \rightarrow q_1 \triangleleft \cdots \triangleleft q_k$.

- A map $\varphi: p \rightarrow q \triangleleft r$ is an element of

$$\varphi \in \mathbf{Poly}(p, q \triangleleft r) \cong \prod_{i \in p(1)} \sum_{j \in q(1)} \prod_{e \in q[j]} \sum_{k \in r(1)} \prod_{f \in r[k]} \sum_{d \in p[i]} 1.$$

We could write it with our abacus pictures:



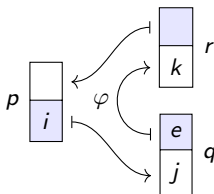
Maps to composites

The abacus pictures are most useful for maps $p \rightarrow q_1 \triangleleft \cdots \triangleleft q_k$.

- A map $\varphi: p \rightarrow q \triangleleft r$ is an element of

$$\varphi \in \mathbf{Poly}(p, q \triangleleft r) \cong \prod_{i \in p(1)} \sum_{j \in q(1)} \prod_{e \in q[j]} \sum_{k \in r(1)} \prod_{f \in r[k]} \sum_{d \in p[i]} 1.$$

We could write it with our abacus pictures:



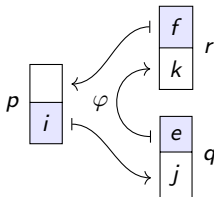
Maps to composites

The abacus pictures are most useful for maps $p \rightarrow q_1 \triangleleft \cdots \triangleleft q_k$.

- A map $\varphi: p \rightarrow q \triangleleft r$ is an element of

$$\varphi \in \mathbf{Poly}(p, q \triangleleft r) \cong \prod_{i \in p(1)} \sum_{j \in q(1)} \prod_{e \in q[j]} \sum_{k \in r(1)} \prod_{f \in r[k]} \sum_{d \in p[i]} 1.$$

We could write it with our abacus pictures:



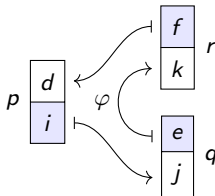
Maps to composites

The abacus pictures are most useful for maps $p \rightarrow q_1 \triangleleft \cdots \triangleleft q_k$.

- A map $\varphi: p \rightarrow q \triangleleft r$ is an element of

$$\varphi \in \mathbf{Poly}(p, q \triangleleft r) \cong \prod_{i \in p(1)} \sum_{j \in q(1)} \prod_{e \in q[j]} \sum_{k \in r(1)} \prod_{f \in r[k]} \sum_{d \in p[i]} 1.$$

We could write it with our abacus pictures:



These will come in handy when asking if two such φ, ψ are equal.

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*,
 - $\epsilon: m \rightarrow I$ is a map, the *counit*, and
 - $\delta: m \rightarrow m \otimes m$ is a map, the *comultiplication*.

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

¹Ahman-Uustalu. “Directed Containers as Categories”. *MSFP 2016*.

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*,
 - $\epsilon: m \rightarrow I$ is a map, the *counit*, and
 - $\delta: m \rightarrow m \otimes m$ is a map, the *comultiplication*.

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

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

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

where $\mathcal{C}[i]$ is the set of morphisms in \mathcal{C} that emanate from i .

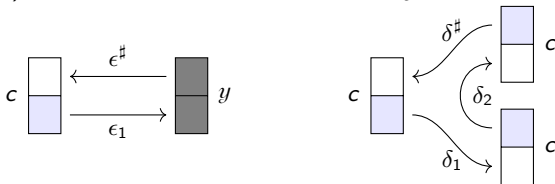
- The counit $\epsilon: \mathfrak{c} \rightarrow y$ assigns to each object an identity.
- The comult $\delta: \mathfrak{c} \rightarrow \mathfrak{c} \triangleleft \mathfrak{c}$ assigns codomains and composites.

¹Ahman-Uustalu. “Directed Containers as Categories”. *MSFP 2016*.

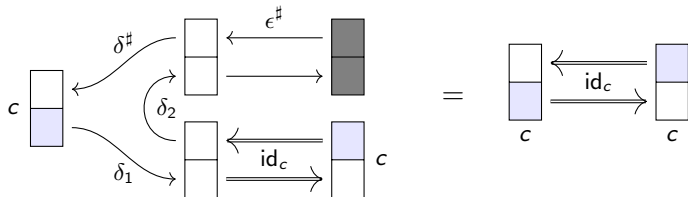
The abacus in action

We can understand the Ahman-Uustalu result combinatorially.

- Let (c, ϵ, δ) be a comonoid, where $\epsilon: c \rightarrow y$ and $\delta: c \rightarrow c \triangleleft c$.



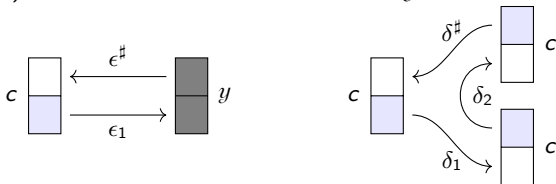
Here's the first unitality law, $(\text{id}_c \triangleleft \epsilon) \circ \delta = \text{id}_c$:



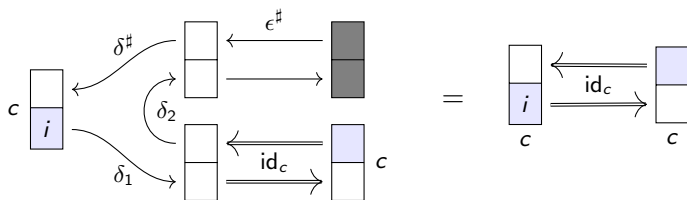
The abacus in action

We can understand the Ahman-Uustalu result combinatorially.

- Let (c, ϵ, δ) be a comonoid, where $\epsilon: c \rightarrow y$ and $\delta: c \rightarrow c \triangleleft c$.



Here's the first unitality law, $(\text{id}_c \triangleleft \epsilon) \circ \delta = \text{id}_c$:

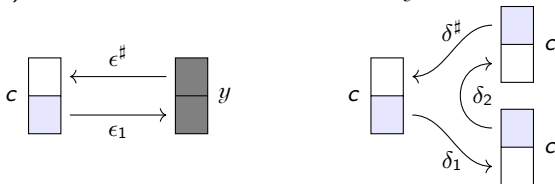


Equation: $\forall i \in c(1) \dots$

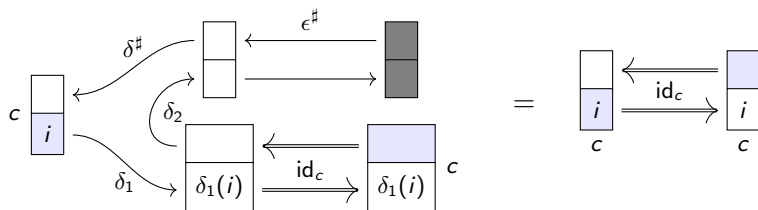
The abacus in action

We can understand the Ahman-Uustalu result combinatorially.

- Let (c, ϵ, δ) be a comonoid, where $\epsilon: c \rightarrow y$ and $\delta: c \rightarrow c \triangleleft c$.



Here's the first unitality law, $(\text{id}_c \triangleleft \epsilon) \circ \delta = \text{id}_c$:

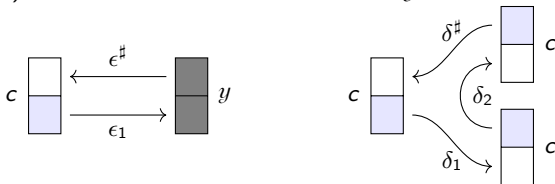


Equation: $\forall i \in c(1), \delta_1(i) = i \wedge \dots$

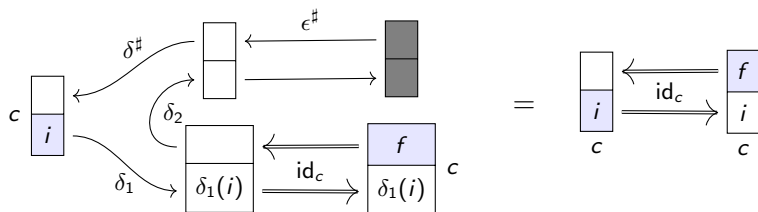
The abacus in action

We can understand the Ahman-Uustalu result combinatorially.

- Let (c, ϵ, δ) be a comonoid, where $\epsilon: c \rightarrow y$ and $\delta: c \rightarrow c \triangleleft c$.



Here's the first unitality law, $(\text{id}_c \triangleleft \epsilon) \circ \delta = \text{id}_c$:

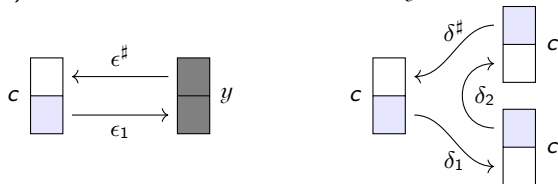


Equation: $\forall i \in c(1), \delta_1(i) = i \wedge \forall f \in c[i], \dots$

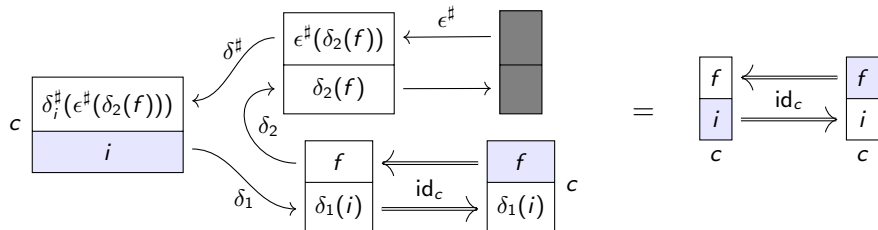
The abacus in action

We can understand the Ahman-Uustalu result combinatorially.

- Let (c, ϵ, δ) be a comonoid, where $\epsilon: c \rightarrow y$ and $\delta: c \rightarrow c \triangleleft c$.



Here's the first unitality law, $(\text{id}_c \triangleleft \epsilon) \circ \delta = \text{id}_c$:

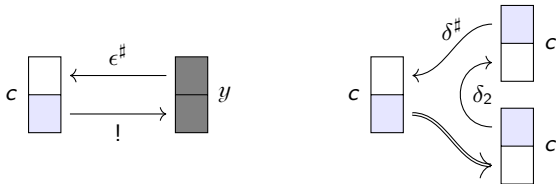


Equation: $\forall i \in c(1), \delta_1(i) = i \wedge \forall f \in c[i], \delta_i^\#(f, \epsilon^\#(\delta_2(f))) = f$.

Making sense of the results

We want to make sense of the set-theoretic equations from the abacus.

- For example, we found out that $\delta_1(i) = i$ for all $i \in c(1)$, so...

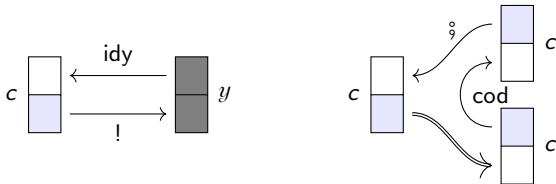


- To make sense of the other equations, let's rename ϵ^\sharp , δ_2 , and δ^\sharp .

Making sense of the results

We want to make sense of the set-theoretic equations from the abacus.

- For example, we found out that $\delta_1(i) = i$ for all $i \in c(1)$, so...

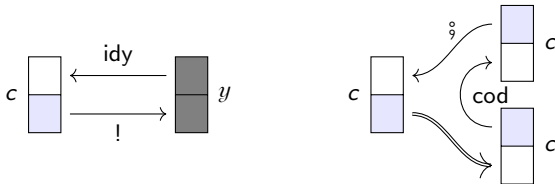


- To make sense of the other equations, let's rename ϵ^\sharp , δ_2 , and δ^\sharp .
- Namely, let's write $\text{idy} := \epsilon^\sharp$, $\text{cod} := \delta_2$, and $\circ := \delta^\sharp$.
 - Then the previous equation says: $f \circ \text{idy}(\text{cod}(f)) = f$.

Making sense of the results

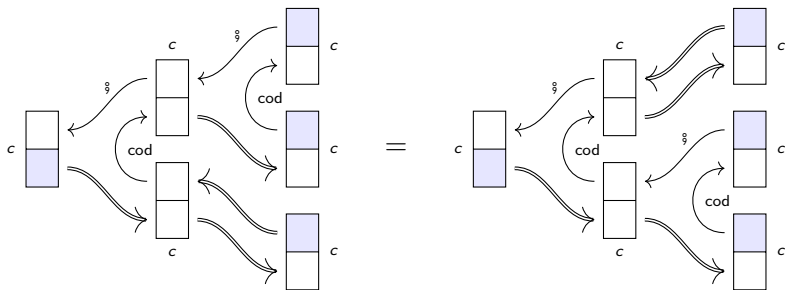
We want to make sense of the set-theoretic equations from the abacus.

- For example, we found out that $\delta_1(i) = i$ for all $i \in c(1)$, so...



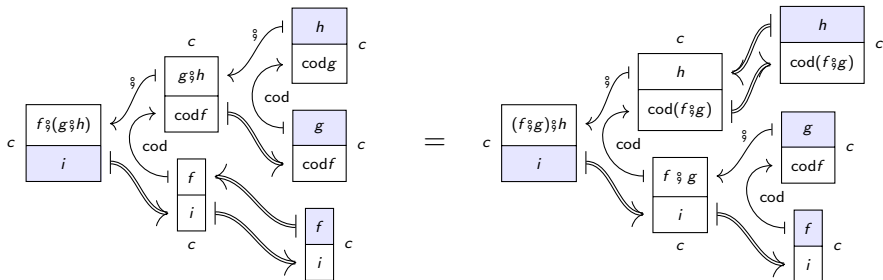
- To make sense of the other equations, let's rename ϵ^\sharp , δ_2 , and δ^\sharp .
- Namely, let's write $\text{idy} := \epsilon^\sharp$, $\text{cod} := \delta_2$, and $\circ := \delta^\sharp$.
 - Then the previous equation says: $f \circ \text{idy}(\text{cod}(f)) = f$.
 - The other unitality eq'n gives: $\text{cod}(\text{idy}(i)) = i$ and $\text{idy}(i) \circ f = f$.
 - The associativity eq'n gives: $\text{cod}(f \circ g) = \text{cod}(g)$ and $(f \circ g) \circ h = f \circ (g \circ h)$.

A brief glance at associativity



Let's fill it in and read off the abacus:

A brief glance at associativity



Let's fill it in and read off the abacus:

$$\forall i \in c(1), i = i \wedge$$

$$\forall f \in c[i], \text{cod} f = \text{cod} f \wedge$$

$$\forall g \in c[\text{cod} f], \text{cod} g = \text{cod}(f ; g) \wedge$$

$$\forall h \in c[\text{cod} g], f ; (g ; h) = (f ; g) ; h.$$

Comonoid maps are “cofunctors”

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

- A comonoid morphism $\varphi: \mathcal{C} \rightarrow \mathcal{D}$ 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 $\varphi_i^\sharp(f) \in \mathfrak{c}[i]$.
 - Rules: φ^\sharp preserves ids and comps, and φ_1 preserves cods.
- Denote this by **Cat**[♯] := **Comon(Poly)** = (cat'ys and cofunctors).

Comonoid maps are “cofunctors”

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

- A comonoid morphism $\varphi: \mathcal{C} \rightarrow \mathcal{D}$ 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 $\varphi_i^\sharp(f) \in \mathfrak{c}[i]$.
 - Rules: φ^\sharp preserves ids and comps, and φ_1 preserves cods.
- Denote this by **Cat**[♯] := **Comon(Poly)** = (cat'ys and cofunctors).

Example: what is a cofunctor $\mathcal{C} \xrightarrow{\varphi} \mathcal{Y}^{\mathbb{Q}}$?

- It is trivial on objects $i \in \text{Ob}(\mathcal{C})$. Passing back morphisms gives:
- ... a map $\varphi_i^\sharp(q): i \rightarrow i_{+q}$ emanating from i for each $q \in \mathbb{Q}$, s.t....
- ... $\varphi_i^\sharp(0) = \text{id}_i$, so $i_{+0} = i$, and $\varphi_i^\sharp(q) \circ \varphi_{i_{+q}}^\sharp(q') = \varphi_i^\sharp(q + q')$.

Comonoid maps are “cofunctors”

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

- A comonoid morphism $\varphi: \mathcal{C} \rightharpoonup \mathcal{D}$ 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 $\varphi_i^\sharp(f) \in \mathfrak{c}[i]$.
 - Rules: φ^\sharp preserves ids and comps, and φ_1 preserves cods.
- Denote this by **Cat**[♯] := **Comon(Poly)** = (cat'ys and cofunctors).

Example: what is a cofunctor $\mathcal{C} \xrightarrow{\varphi} \mathcal{Y}^{\mathbb{Q}}$?

- It is trivial on objects $i \in \text{Ob}(\mathcal{C})$. Passing back morphisms gives:
- ... a map $\varphi_i^\sharp(q): i \rightarrow i_{+q}$ emanating from i for each $q \in \mathbb{Q}$, s.t....
- ... $\varphi_i^\sharp(0) = \text{id}_i$, so $i_{+0} = i$, and $\varphi_i^\sharp(q) \circ \varphi_{i_{+q}}^\sharp(q') = \varphi_i^\sharp(q + q')$.

“That’s a strange sort of structure to put on a category!”

- Cofunctors offer a whole new world to explore. Think “vector fields”.
- The natural co-transformations between them are even wilder.

Cat[#]: examples and facts

Here are some examples of the polynomial \mathfrak{c} carrying a category \mathcal{C} .

- \mathfrak{c} never has constant part: every object needs an outgoing arrow.
- $\mathfrak{c} = Oy$ is linear iff \mathcal{C} is a discrete category, with $\text{Ob}(\mathcal{C}) = O$.
- $\mathfrak{c} = y^M$ is representable iff $M \in \mathbf{Set}$ carries a monoid.
- If $\mathcal{C} = \boxed{\overset{1}{\bullet} \rightarrow \overset{2}{\bullet} \rightarrow \cdots \rightarrow \overset{N}{\bullet}}$ then $\mathfrak{c} = y^N + y^{N-1} + \cdots + y$.

Cat[#]: examples and facts

Here are some examples of the polynomial \mathfrak{c} carrying a category \mathcal{C} .

- \mathfrak{c} never has constant part: every object needs an outgoing arrow.
- $\mathfrak{c} = Oy$ is linear iff \mathcal{C} is a discrete category, with $\text{Ob}(\mathcal{C}) = O$.
- $\mathfrak{c} = y^M$ is representable iff $M \in \mathbf{Set}$ carries a monoid.
- If $\mathcal{C} = \boxed{1 \rightarrow 2 \rightarrow \dots \rightarrow N}$ then $\mathfrak{c} = y^N + y^{N-1} + \dots + y$.

Other facts about **Cat[#]**:

- Coproducts in **Cat[#]** and in **Cat** agree; carrier is $\mathfrak{c} + \mathfrak{d}$.
- **Cat[#]** has finite products (Niu), and they're very interesting.
- **Cat[#]** inherits \otimes from **Poly**, and $\mathfrak{c} \otimes \mathfrak{d}$ is the usual categorical product.

Cofree comonoids

To any polynomial p , we can associate the *cofree comonoid* on p .

- That is, the forgetful functor $\mathbf{Cat}^\sharp \rightarrow \mathbf{Poly}$ has a right adjoint.
- I'll give an explicit description on the next slide.
- There's a standard construction for this type of thing.

We need a polynomial \mathfrak{c}_p and maps $\mathfrak{c}_p \rightarrow y$ and $\mathfrak{c}_p \rightarrow \mathfrak{c}_p \triangleleft \mathfrak{c}_p$.

Cofree comonoids

To any polynomial p , we can associate the *cofree comonoid* on p .

- That is, the forgetful functor $\mathbf{Cat}^\sharp \rightarrow \mathbf{Poly}$ has a right adjoint.
- I'll give an explicit description on the next slide.
- There's a standard construction for this type of thing.

We need a polynomial c_p and maps $c_p \rightarrow y$ and $c_p \rightarrow c_p \triangleleft c_p$.

- Starting with $p \in \mathbf{Poly}$, we first copoint it by multiplying by y .
- That is, py is the universal thing mapping to p and y .
- We get c_p by taking the limit of the following diagram in \mathbf{Poly} :

$$c_p := \lim \left(y \longleftarrow py \xleftarrow{\quad} py \triangleleft py \xleftarrow{\quad} py \triangleleft py \triangleleft py \xleftarrow{\quad} \cdots \right)$$

Cofree comonoids

To any polynomial p , we can associate the *cofree comonoid* on p .

- That is, the forgetful functor $\mathbf{Cat}^\# \rightarrow \mathbf{Poly}$ has a right adjoint.
- I'll give an explicit description on the next slide.
- There's a standard construction for this type of thing.

We need a polynomial c_p and maps $c_p \rightarrow y$ and $c_p \rightarrow c_p \triangleleft c_p$.

- Starting with $p \in \mathbf{Poly}$, we first copoint it by multiplying by y .
- That is, py is the universal thing mapping to p and y .
- We get c_p by taking the limit of the following diagram in \mathbf{Poly} :

$$c_p := \lim \left(y \longleftarrow py \xleftarrow{\quad} py \triangleleft py \xleftarrow{\quad} py \triangleleft py \triangleleft py \xleftarrow{\quad} \cdots \right)$$

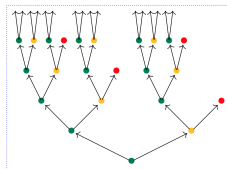
For us, a main use of c_p is an equivalence $c_p\text{-}\mathbf{Set} \cong p\text{-}\mathbf{Coalg}$.

- A coalgebra $S \rightarrow p(S)$ corresponds to $c_p \rightarrow \mathbf{Set}$ with elements S .
- For example, the object set $c_p(1)$ is the terminal p -coalgebra.

The cofree comonoid \mathfrak{c}_p via p -trees

Comonoids in **Poly** are categories, so \mathbf{c}_p is a category; which one?

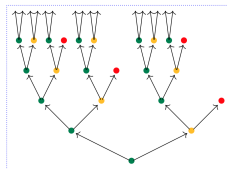
- It's actually free on a graph, but the graph is very interesting.
- The vertex-set $\mathfrak{c}_p(1)$ of the graph is the set of p -trees.
 - A p -tree is a possibly infinite tree t , where each node...
 - ...is labeled by a position $i \in p(1)$ and has $p[i]$ -many branches.
 - Example object $t \in \mathfrak{c}_p(1)$, where $p = \{\bullet, \bullet, \bullet\}y^2 + \{\bullet\} \cong 2y^2 + 1$:



The cofree comonoid \mathfrak{c}_p via p -trees

Comonoids in **Poly** are categories, so \mathbf{c}_p is a category; which one?

- It's actually free on a graph, but the graph is very interesting.
- The vertex-set $\mathfrak{c}_p(1)$ of the graph is the set of p -trees.
 - A p -tree is a possibly infinite tree t , where each node...
 - ...is labeled by a position $i \in p(1)$ and has $p[i]$ -many branches.
 - Example object $t \in \mathfrak{c}_p(1)$, where $p = \{\bullet, \bullet, \bullet\}y^2 + \{\bullet\} \cong 2y^2 + 1$:



- For any vertex $t \in \mathfrak{c}_p(1)$, an arrow $a \in \mathfrak{c}_p[t]$ emanating from t is...
- ...a finite path from the root of t to another node in t .
- Its codomain is the p -tree sitting at the target node (its root).
- Identity arrow = length-0 path; composition = path concatenation.

Imagine the whole graph c_p : every possible “destiny” is included.

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

categories

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

- It's a polynomial m , equipped with two morphisms in **Poly**

$$\mathfrak{c} \triangleleft m \xleftarrow{\lambda} m \xrightarrow{\rho} m \triangleleft \mathfrak{d}$$

each cohering naturally with the comonoid structure ϵ, δ for $\mathfrak{c}, \mathfrak{d}$.

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

categories

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

- It's a polynomial m , equipped with two morphisms in **Poly**

$$c \triangleleft m \xleftarrow{\lambda} m \xrightarrow{\rho} m \triangleleft d$$

each cohering naturally with the comonoid structure ϵ, δ for c, d .

- I denote this $(\mathcal{C}, \mathcal{D})$ -bicomodule m like so:

$$c \triangleleft^m d \quad \text{or} \quad \mathcal{C} \triangleleft^m \mathcal{D}$$

- 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 explained² that bicomodules $m \in {}_C \mathbf{Mod}_{\mathcal{D}}$, which we've denoted

$$C \triangleleft^m \triangleleft \mathcal{D} \quad \text{or} \quad \mathfrak{c} \triangleleft^m \triangleleft \mathfrak{d}$$

can be identified with parametric right adjoint functors (prafunctors)

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

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

Bicomodules are parametric right adjoints

Garner explained² that bicomodules $m \in {}_C\mathbf{Mod}_{\mathcal{D}}$, which we've denoted

$$C \triangleleft^m \triangleleft \mathcal{D} \quad \text{or} \quad \mathfrak{c} \triangleleft^m \triangleleft \mathfrak{d}$$

can be identified with parametric right adjoint functors (prafunctors)

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

- From this perspective the arrow points in the expected direction.
- Assuming Garner's result, check: ${}_C\mathbf{Mod}_0 \cong C\text{-}\mathbf{Set}$.

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

Bicomodules are parametric right adjoints

Garner explained² that bicomodules $m \in {}_C \mathbf{Mod}_{\mathcal{D}}$, which we've denoted

$$C \triangleleft \xrightarrow{m} \triangleleft \mathcal{D} \quad \text{or} \quad \mathfrak{c} \triangleleft \xrightarrow{m} \triangleleft \mathfrak{d}$$

can be identified with parametric right adjoint functors (prafunctors)

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

- From this perspective the arrow points in the expected direction.
- Assuming Garner's result, check: ${}_C \mathbf{Mod}_0 \cong C\text{-}\mathbf{Set}$.

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

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

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

Let's ask the abacus

To prove that bicomodules $\mathfrak{c} \triangleleft^m \triangleleft \mathfrak{d}$ are prafunctors ${}_{\mathfrak{d}}\mathbf{Mod}_0 \rightarrow {}_{\mathfrak{c}}\mathbf{Mod}_0$:

- Write out the bicomodule equations and run the abacus.

The diagrammatic proof shows the equality of two expressions. The left expression is a node m with two outgoing arrows to a node d (labeled ρ) and a node m (labeled ϵ). The right expression is a node m with two outgoing arrows to a node m (labeled id) and a node m (labeled id). The word "and" is placed between the two expressions. The next part of the proof shows the equality of two more expressions. The left expression is a node m with two outgoing arrows to a node d (labeled ρ) and a node m (labeled ρ). The right expression is a node m with two outgoing arrows to a node d (labeled δ) and a node m (labeled id).

The diagrammatic proof shows the Frobenius property for the comultiplication map c . It consists of two parts, separated by the word "and".

Left part: A diagrammatic equation showing that the comultiplication map c is a homomorphism for the multiplication map m . On the left, a multiplication map m (represented by two parallel lines with a curved arrow labeled λ) is followed by a comultiplication map c (represented by two parallel lines with a curved arrow labeled ϵ). This is equal to a single multiplication map m (represented by two parallel lines with a curved arrow labeled id).

Right part: A diagrammatic equation showing that the comultiplication map c is a homomorphism for the comultiplication map c . On the left, a comultiplication map c (represented by two parallel lines with a curved arrow labeled δ) is followed by another comultiplication map c (represented by two parallel lines with a curved arrow labeled ϵ). This is equal to a single comultiplication map c (represented by two parallel lines with a curved arrow labeled id).

Diagrammatic equation for the associativity of the multiplication map m . The left side shows a multiplication map m from two copies of d to c , with an intermediate multiplication map m from d to m . The right side shows a multiplication map m from two copies of m to c , with an intermediate multiplication map m from m to m . The two diagrams are equated.

Interpreting the abacus

By running the abacus and interpreting the results, we find the following.

- A left comodule $\mathfrak{c} \triangleleft m \xleftarrow{\lambda} m$ can be identified with a functor $\mathfrak{c} \rightarrow \mathbf{Poly}$.

$$m \cong \sum_{i \in \mathfrak{c}(1)} \sum_{x \in m_i} y^{m[x]}$$

- The right comodule conditions on $m \xrightarrow{\rho} m \triangleleft d$ say that each $m[x] \dots$
- ... is not just a set, it's the set of elements for a copresheaf on \mathfrak{d} !

Interpreting the abacus

By running the abacus and interpreting the results, we find the following.

- A left comodule $\mathfrak{c} \triangleleft m \xleftarrow{\lambda} m$ can be identified with a functor $\mathfrak{c} \rightarrow \mathbf{Poly}$.

$$m \cong \sum_{i \in \mathfrak{c}(1)} \sum_{x \in m_i} y^{m[x]}$$

- The right comodule conditions on $m \xrightarrow{\rho} m \triangleleft d$ say that each $m[x] \dots$
- ... is not just a set, it's the set of elements for a copresheaf on \mathfrak{d} !

When we add the coherence condition, it all falls into place.

- The idea is that each $i \in \mathfrak{c}(1)$ functorially gets a set m_i and...
- ... each $x \in m_i$ gets a \mathfrak{d} -set with elements $m[x]$.
- The prafunctor $\mathfrak{d}\text{-}\mathbf{Set} \rightarrow \mathfrak{c}\text{-}\mathbf{Set}$ associated to m takes any \mathfrak{d} -set N , ...
- ... hom's in the $m[x]$'s, and adds them up to get a \mathfrak{c} -set.

We'll understand this better semantically when we get to applications.

Getting acquainted with bicomodules

Here are some facts, just to get you acquainted with $\mathfrak{c} \triangleleft^m \triangleright \mathfrak{d}$.

- If $\mathfrak{d} = 0$ then carrier $m \in \mathbf{Poly}$ is constant, i.e. $m = M$ for $M \in \mathbf{Set}$.
- If carrier $m = M$ is constant, then m factors as $\mathfrak{c} \triangleleft^M \triangleright 0 \triangleleft^! \triangleright \mathfrak{d}$.

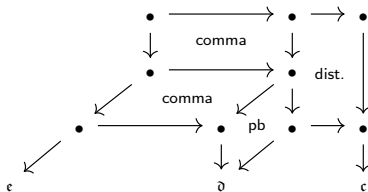
Getting acquainted with bicomodules

Here are some facts, just to get you acquainted with $\mathfrak{c} \triangleleft^m \triangleleft \mathfrak{d}$.

- If $\mathfrak{d} = 0$ then carrier $m \in \mathbf{Poly}$ is constant, i.e. $m = M$ for $M \in \mathbf{Set}$.
- If carrier $m = M$ is constant, then m factors as $\mathfrak{c} \triangleleft^M \triangleleft 0 \triangleleft^! \triangleleft \mathfrak{d}$.
- The following cat'ies are isomorphic and all are equivalent to $\mathfrak{c}\text{-}\mathbf{Set}$:
 - Cartesian cofunctors over \mathfrak{c} = Discrete opfibrations over \mathfrak{c} .
 - The **constant left** \mathfrak{c} -comodules, i.e. with constant carrier $m = M$.
 - The **linear left** \mathfrak{c} -comodules, i.e. with linear carrier $m = My$.
 - The **representable right** \mathfrak{c} -comodules, i.e. with carrier y^M .

Bicomodule composition

If you've ever tried to compose prafunctors; this might look familiar.



But in **Poly**, it's just given by the usual bicomodule composition.

- The composite of $c \triangleleft^m d \triangleleft^n e$, is carried by the equalizer:

$$m \triangleleft_d n \xrightarrow{eq} m \triangleleft n \rightrightarrows m \triangleleft d \triangleleft n$$

- This has a natural (c, e) -structure, because \triangleleft preserves conn. limits.
- It's amazing to see the combinatorics handle all this complexity.

The framed bicategory \mathbb{P}

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

$$\begin{array}{ccc}
 \mathfrak{c} & \xleftarrow{m} & \mathfrak{d} \\
 \varphi \downarrow & \Downarrow \alpha & \downarrow \psi \\
 \mathfrak{c}' & \xleftarrow{m'} & \mathfrak{d}'
 \end{array}$$

- It's got a ton of structure, e.g. two monoidal structures, $+$, \otimes .
- It's actually not too hard to describe.

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

- ${}_c\mathbf{Mod}_0 \cong \mathcal{C}\text{-}\mathbf{Set}$, copresheaves on \mathcal{C} .
- ${}_1\mathbf{Mod}_{\mathcal{D}} \cong \mathbf{Coco}((\mathcal{D}\text{-}\mathbf{Set})^{\mathrm{op}})$.
- ${}_c\mathbf{Mod}_{\mathcal{D}} \cong \mathbf{Cat}(\mathcal{C}, {}_1\mathbf{Mod}_{\mathcal{D}})$.

The framed bicategory \mathbb{P}

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

$$\begin{array}{ccccc}
 \mathfrak{c} & \xleftarrow{m} & \triangleleft & \mathfrak{d} & \\
 \varphi \downarrow & & \Downarrow \alpha & & \downarrow \psi \\
 \mathfrak{c}' & \xleftarrow{m'} & \triangleleft & \mathfrak{d}' &
 \end{array}$$

- It's got a ton of structure, e.g. two monoidal structures, $+$, \otimes .
- It's actually not too hard to describe.

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

- ${}_c\mathbf{Mod}_0 \cong \mathcal{C}\text{-}\mathbf{Set}$, copresheaves on \mathcal{C} .
- ${}_1\mathbf{Mod}_{\mathcal{D}} \cong \mathbf{Coco}((\mathcal{D}\text{-}\mathbf{Set})^{\mathrm{op}})$.
- ${}_c\mathbf{Mod}_{\mathcal{D}} \cong \mathbf{Cat}(\mathcal{C}, {}_1\mathbf{Mod}_{\mathcal{D}})$.

There's a factorization system on \mathbb{P} :

- Every $m \in {}_c\mathbf{Mod}_0$ can be factored as $m \cong f \circ p$,

$$\mathfrak{c} \xleftarrow{f} \triangleleft \mathfrak{c}' \xleftarrow{p} \triangleleft \mathfrak{d}$$

where f “is” a discrete opfibration and p “is” a profunctor.

Gambino-Kock's framed bicategory $\mathbb{P}\mathbf{Poly}$

In Gambino-Kock, the authors construct a framed bicategory $\mathbb{P}\mathbf{poly}_{\mathbf{Set}}$.

- Its vertical category is **Set**.
- A horizontal map $I \rightarrow J$ is J -many polynomials in I -many variables.
- 2-cells are natural transformations between polynomial functors.

Gambino-Kock's framed bicategory $\mathbb{P}\mathbf{poly}$

In Gambino-Kock, the authors construct a framed bicategory $\mathbb{P}\mathbf{poly}_{\mathbf{Set}}$.

- Its vertical category is **Set**.
- A horizontal map $I \rightarrow J$ is J -many polynomials in I -many variables.
- 2-cells are natural transformations between polynomial functors.

This is a full subcategory $\mathbb{P}\mathbf{poly} \subseteq \mathbb{P}$.

- Objects in \mathbb{P} are categories; those in $\mathbb{P}\mathbf{poly}$ are the discrete categories.
- Verticals in \mathbb{P} are cofunctors; $\mathbf{Set}(I, I') \cong \mathbf{Cat}^\#(Iy, I'y)$.
- Horizontals in \mathbb{P} are prafunctors; between discretess, these are poly's.
- In both, 2-cells are the natural transformations.

The comonoid theory \mathbb{P} of (one-variable) **Poly** includes all of $\mathbb{P}\mathbf{poly}$.

Adjunctions in \mathbb{P}

The map $_ \mathbf{Mod}_0: \mathbb{P}^{\text{op}} \rightarrow \mathbb{Cat}$ is locally fully faithful; i.e....

- ...for categories \mathcal{C}, \mathcal{D} , only some functors $m: \mathcal{D}\text{-}\mathbf{Set} \rightarrow \mathcal{C}\text{-}\mathbf{Set}$ count...
- ... as bimodules $\mathcal{C} \xleftarrow{m} \mathcal{D}$, but for those m, n that do...
- ... the bimodule maps $m \Rightarrow n$ are exactly the natural transformations.

Thus it is easy to say when $\mathcal{C} \xleftarrow{m} \mathcal{D}$ has an adjoint in \mathbb{P} , namely if...

- ...the induced $\mathcal{D}\text{-}\mathbf{Set} \xrightarrow{m} \mathcal{C}\text{-}\mathbf{Set}$ has an adjoint $\mathcal{C}\text{-}\mathbf{Set} \xrightarrow{m'} \mathcal{D}\text{-}\mathbf{Set}$ and...
- ... m' is in \mathbb{P} ! (i.e. the adjoint m' needs to preserve connected limits).

Adjunctions in \mathbb{P}

The map $_ \mathbf{Mod}_0: \mathbb{P}^{\text{op}} \rightarrow \mathbf{Cat}$ is locally fully faithful; i.e....

- ...for categories \mathcal{C}, \mathcal{D} , only some functors $m: \mathcal{D}\text{-}\mathbf{Set} \rightarrow \mathcal{C}\text{-}\mathbf{Set}$ count...
- ... as bimodules $\mathcal{C} \xleftarrow{m} \mathcal{D}$, but for those m, n that do...
- ... the bimodule maps $m \Rightarrow n$ are exactly the natural transformations.

Thus it is easy to say when $\mathcal{C} \xleftarrow{m} \mathcal{D}$ has an adjoint in \mathbb{P} , namely if...

- ...the induced $\mathcal{D}\text{-}\mathbf{Set} \xrightarrow{m} \mathcal{C}\text{-}\mathbf{Set}$ has an adjoint $\mathcal{C}\text{-}\mathbf{Set} \xrightarrow{m'} \mathcal{D}\text{-}\mathbf{Set}$ and...
- ... m' is in \mathbb{P} ! (i.e. the adjoint m' needs to preserve connected limits).

Both functors $\mathcal{C} \xrightarrow{F} \mathcal{D}$ and cofunctors $\mathcal{C} \xrightarrow{\varphi} \mathcal{D}$ induce adjunctions in \mathbb{P}^{op} .

- The pullback and right Kan extension along F are adjoint $\Delta_F \dashv \Pi_F$.
- The companion and conjoint of φ are adjoint $\Sigma_\varphi \dashv \Delta_\varphi$.
- A dopf F is both a functor and a cofunctor, and the Δ 's coincide.

Note that cofunctors $\mathcal{C} \rightarrow \mathcal{D}$ induce interesting maps between toposes:

- Whereas geometric morphisms $\mathcal{C}\text{-}\mathbf{Set} \rightleftarrows \mathcal{D}\text{-}\mathbf{Set}$ preserve finite limits...
- ... cofunctors induce adjunctions that preserve connected limits.

Operads as monads in \mathbb{P}

In any framed bicategory, notation from \mathbb{P} , a monad (C, m, η, μ) consists of

- An object C , the *type*
- a bicomodule $C \xleftarrow{m} \triangleleft C$, the *carrier*
- a 2-cell $\eta: \text{id}_C \Rightarrow m$, the *unit*
- a 2-cell $\mu: m \circ m \Rightarrow m$, the *multiplication*
- satisfying the usual laws.

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 bicomodule $\mathcal{C} \xleftarrow{m} \mathcal{C}$, the *carrier*
- a 2-cell $\eta: \text{id}_{\mathcal{C}} \Rightarrow m$, the *unit*
- a 2-cell $\mu: m \circ m \Rightarrow m$, the *multiplication*
- satisfying the usual laws.

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

- When $\mathcal{C} \cong I$ is discrete, η, μ are cartesian, you get colored operads.³
- Relaxing discreteness of \mathcal{C} , the domain of a morphism can be...
- ... a diagram, rather than a mere set, of objects.
- Relaxing “iso” condition, composites and ids can have “weird” arities.

³Not quite the standard definition of operad, but no less elegant: the input to a morphism is a set, rather than a list of objects. You can also talk about standard (list-based) operads and their generalizations within the \mathbb{P} setting; see Gambino-Kock.

“Categories = monads in \mathbf{Span} ” in \mathbb{P}

It is well-known that “categories are monads in \mathbf{Span} .” Let O be a set.

- A profunctor $Oy \triangleright \xrightarrow{m} \triangleleft Oy$ acts as a span iff it's a left adjoint.
- If a monad m has a right adjoint $Oy \triangleleft \xleftarrow{c} \triangleleft Oy$, then c is a comonad.
- Now, since the vertical part of \mathbb{P} is already **Comon(Poly)**,
- ... c has a canonical comonoid structure \mathfrak{c} , equipped with $\mathfrak{c} \rightarrowtail Oy$.
- This map $\mathfrak{c} \rightarrowtail Oy$ is identity on objects because c was right adjoint.
- Thus we see internally how m induces a category \mathfrak{c} with object-set O .

“Categories = monads in $\mathbb{S}\text{pan}$ ” in \mathbb{P}

It is well-known that “categories are monads in $\mathbb{S}\text{pan}$.” Let O be a set.

- A prafunctor $Oy \triangleright \xrightarrow{m} \triangleright Oy$ acts as a span iff it's a left adjoint.
- If a monad m has a right adjoint $Oy \triangleleft \xleftarrow{c} \triangleleft Oy$, then c is a comonad.
- Now, since the vertical part of \mathbb{P} is already **Comon(Poly)**,
- ... c has a canonical comonoid structure \mathfrak{c} , equipped with $\mathfrak{c} \rightarrowtail Oy$.
- This map $\mathfrak{c} \rightarrowtail Oy$ is identity on objects because c was right adjoint.
- Thus we see internally how m induces a category \mathfrak{c} with object-set O .

Here's how functors and cofunctors look in this perspective:

$$\begin{array}{ccc}
 Oy \triangleright \xrightarrow{m} \triangleright Oy & & Oy \triangleleft \xleftarrow{c} \triangleleft Oy \\
 \downarrow & \Downarrow & \downarrow \\
 O'y \triangleright \xrightarrow{m'} \triangleright O'y & \text{vs.} & O'y \triangleleft \xleftarrow{c'} \triangleleft O'y
 \end{array}$$

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 gives formula for gluing, but no uniqueness guarantee.

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 gives formula for gluing, but no uniqueness guarantee.

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

- 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 $m \in \mathbf{Poly}$:

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

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

Grothendieck sites give \mathbb{P} -monads

Every Grothendieck site $(\mathcal{C}^{\text{op}}, J)$ 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 gives formula for gluing, but no uniqueness guarantee.

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

- 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 $m \in \mathbf{Poly}$:

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

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

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.

$$\begin{array}{ccccc} \mathcal{C} & \xleftarrow{m} & \mathcal{C} & \xleftarrow{P} & 0 \\ & \swarrow & \downarrow h & \searrow & \\ & & P & & \end{array}$$

Outline

1 Introduction

2 Theory

3 Applications

- Interacting Moore machines
- Mode-dependence
- Databases
- Cellular automata
- Deep learning

4 Conclusion

Bringing the abacus out of the monastery

I hope it's now clear that we've got a well-oiled machine:

- **Poly** and \mathbb{P} have excellent formal properties, and
- we can see how they work using very concrete calculations.

Our next job is to take this shiny abacus out for a spin.

- How do I see **Poly** as appropriate for the Glass Bead Game?
- We can use this instrument to talk about many aspects of the world.

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 $r: S \rightarrow B$, called *readout*, and
- a function $u: S \times A \rightarrow S$, 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* 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 $r: S \rightarrow B$, called *readout*, and
- a function $u: S \times A \rightarrow S$, 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* of the Moore machine.



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

- Given a list/stream $[a_0, a_1, \dots]$ 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 $[b_0, b_1, \dots]$ of B 's.

Moore machines as maps in Poly

We can understand Moore machines $A \boxed{S} B$ in terms of polynomials.

- A Moore machine $r: S \rightarrow B$ and $u: S \times A \rightarrow S$ is:
 - A function $S \rightarrow B \times S^A$, i.e. a By^A -coalgebra.
 - (It can also be phrased as a polynomial map $Sy^S \rightarrow By^A$.)

Moore machines as maps in **Poly**

We can understand Moore machines $A \boxed{S}^B$ in terms of polynomials.

- A Moore machine $r: S \rightarrow B$ and $u: S \times A \rightarrow S$ is:
 - A function $S \rightarrow B \times S^A$, i.e. a By^A -coalgebra.
 - (It can also be phrased as a polynomial map $Sy^S \rightarrow By^A$.)

A *p-coalgebra* allows different input-sets at different positions.

- For arbitrary $p \in \mathbf{Poly}$ we can interpret a map $\varphi: S \rightarrow p \triangleleft S$ 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 $\varphi_2(s, d) \in S$.

Moore machines as maps in Poly

We can understand Moore machines $A \boxed{S} B$ in terms of polynomials.

- A Moore machine $r: S \rightarrow B$ and $u: S \times A \rightarrow S$ is:
 - A function $S \rightarrow B \times S^A$, i.e. a By^A -coalgebra.
 - (It can also be phrased as a polynomial map $Sy^S \rightarrow By^A$.)

A p -coalgebra allows different input-sets at different positions.

- For arbitrary $p \in \mathbf{Poly}$ we can interpret a map $\varphi: S \rightarrow p \triangleleft S$ 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 $\varphi_2(s, d) \in S$.

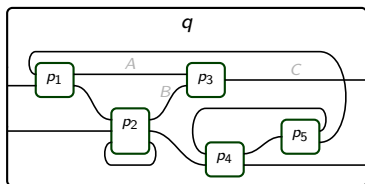
Even more general: a functor $S: \mathcal{C} \rightarrow \mathbf{Set}$ for any category \mathcal{C} .

- This generalizes the above, because $p\text{-Coalg} \cong \mathbf{c}_p\text{-Set}$.
- Imagine its elements (c, s) as states; each reads out its object $c \in \mathcal{C} \dots$
- ... and for any morphism $f: c \rightarrow c'$, it can be updated to $(c', s.f)$.

We'll call any of these things *dynamical systems*.

Wiring diagrams

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



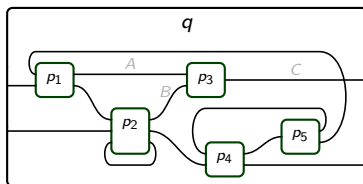
(φ)

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 $\varphi: p_1 \otimes \cdots \otimes p_5 \rightarrow q$.
 - 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.

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 $\varphi: p_1 \otimes \cdots \otimes p_5 \rightarrow q$.
 - 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.
- Now each subsystem can be endowed with a coalgebra $S_i \rightarrow p_i \triangleleft S_i$.
- We tensor and compose to give $S \rightarrow q \triangleleft S$, where $S := S_1 \times \cdots \times S_5$.

So φ applied to dynamics in p_1, \dots, p_5 gives dynamics in q .

More general interaction



The whole picture above 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 $Wy \otimes Wy \otimes 2y^W \rightarrow y$.

- That's a map $2W^2y^W \rightarrow y$.
- Equivalently, it's a function $2W^2 \rightarrow W$. 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 functor $S: T \rightarrow \mathbf{Set}$, as in our general definition above.

Other sorts of dynamical systems

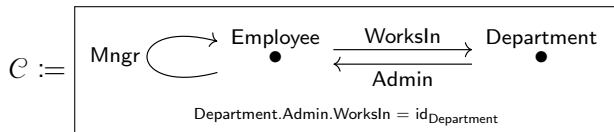
Dynamical systems are usually defined as actions of a monoid T .

- 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 functor $S: T \rightarrow \mathbf{Set}$, as in our general definition above.

Summary: **Poly** can encode dynamical systems and rewiring diagrams.

Categorical databases

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



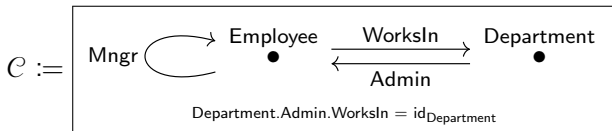
A functor $I: \mathcal{C} \rightarrow \mathbf{Set}$ (i.e. $\mathcal{C} \xleftarrow{I} 0$) can be represented as follows:

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.



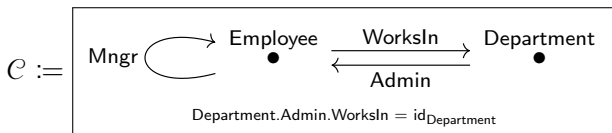
A functor $I: \mathcal{C} \rightarrow \mathbf{Set}$ (i.e. $\mathcal{C} \xleftarrow{I} 0$) can be represented as follows:

Employee	WorksIn	Mngr	Department	Admin
♥	P9	♥	bLue	T****
T****	bLue	orca	P9	♥
orca	bLue	orca		

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

Categorical databases

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



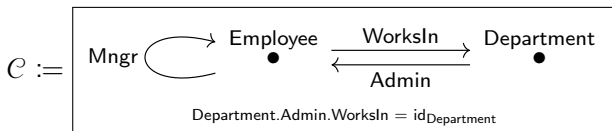
More realistically, data should include *attributes* and look like this:

Employee	FName	WorksIn	Mngr
♥	Alan	P9	♥
T****	Dani	bLue	orca
orca	Sara	bLue	orca

Department	DName	Secr
bLue	Sales	T****
P9	IT	♥

Categorical databases

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



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

Employee	FName	WorksIn	Mngr
♥	Alan	P9	♥
T****	Dani	bLue	orca
orca	Sara	bLue	orca

Department	DName	Secr
bLue	Sales	T****
P9	IT	♥

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

$$\begin{array}{ccc} \mathcal{C} & \xleftarrow{I} \triangleleft & 0 \\ \downarrow & \Downarrow \alpha & \parallel \\ \text{Ob}(\mathcal{C}) & \xleftarrow{T} \triangleleft & 0 \end{array}$$

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 ${}_c\mathbf{Mod}_{\mathcal{D}}$ can be understood as follows.

- First, it's a functor $\mathcal{C} \rightarrow {}_1\mathbf{Mod}_{\mathcal{D}}$, so what's an object in ${}_1\mathbf{Mod}_{\mathcal{D}}$?
- 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 ${}_c\mathbf{Mod}_{\mathcal{D}}$ can be understood as follows.

- First, it's a functor $\mathcal{C} \rightarrow {}_1\mathbf{Mod}_{\mathcal{D}}$, so what's an object in ${}_1\mathbf{Mod}_{\mathcal{D}}$?
- 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 $\mathcal{D} = \left(\begin{array}{ccc} \text{City} & \xrightarrow{\text{in}} & \text{State} \xleftarrow{\text{in}} \text{County} \\ \bullet & & \bullet \end{array} \right)$, a duc-query might be...

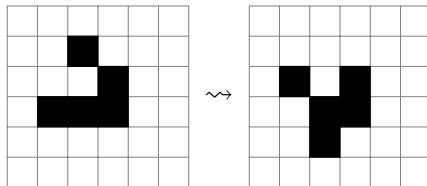
$$(\text{City} \times_{\text{State}} \text{City}) + (\text{City} \times_{\text{State}} \text{County}) + (\text{County} \times_{\text{State}} \text{County})$$

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

Cellular automata

Cellular automata are like Moore machines, except with no internal state.

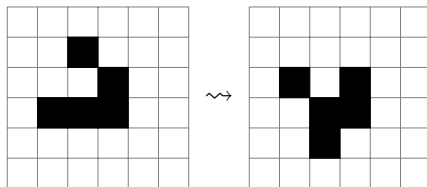
- Here's a picture of a *glider* from Conway's Game of Life:



Cellular automata

Cellular automata are like Moore machines, except with no internal state.

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

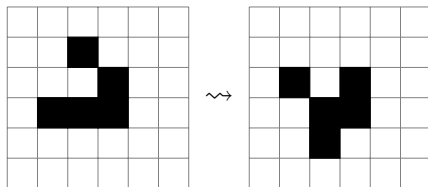


- GoL takes place on a grid, a set $V := \mathbb{Z} \times \mathbb{Z}$ of “squares”
- Each square has neighbors; think of the grid as a graph $A \Rightarrow V$.
- Each square can be in one of two states: white or black.

Cellular automata

Cellular automata are like Moore machines, except with no internal state.

- 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 $A \Rightarrow V$.
- 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 $Vy \triangleleft \xrightarrow{g} \triangleleft Vy$
 - Each $v \in V$ queries its neighbors (and itself).
 - The carrier of the prafunctor for GoL is $g := Vy^9$.
 - In fact, g 's a profunctor: it preserves the terminal, $(g \circ V) \cong V$.

Cellular automata as algebras in \mathbb{P}

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

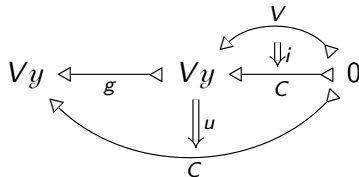
- We encode the graph $A \Rightarrow V$ as a prafunctor $Vy \xleftarrow{g} Vy$
 - Each $v \in V$ queries its neighbors (and itself).
 - The carrier of the prafunctor for GoL is $g := Vy^9$.
 - In fact, g 's a profunctor: it preserves the terminal, $(g \circ V) \cong V$.
- We encode the color-set for 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

$$\begin{array}{ccccc}
 Vy & \xleftarrow{g} & Vy & \xleftarrow{C} & 0 \\
 & & \Downarrow u & & \\
 & \curvearrowright & & \curvearrowleft & \\
 & & C & &
 \end{array}$$

Cellular automata as algebras in \mathbb{P}

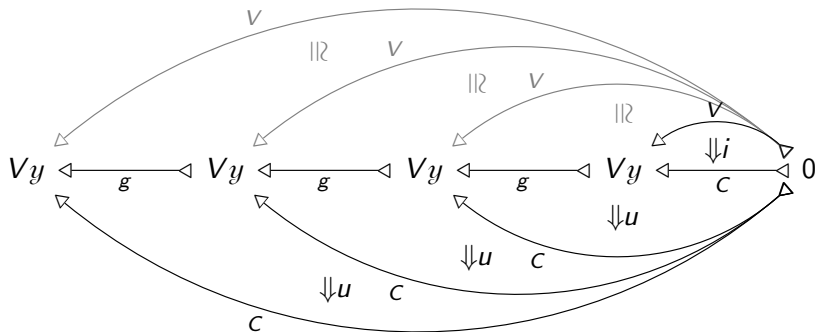
How do we encode this in \mathbb{P} ?

- We encode the graph $A \Rightarrow V$ as a prafunctor $Vy \xleftarrow{g} Vy$
 - Each $v \in V$ queries its neighbors (and itself).
 - The carrier of the prafunctor for GoL is $g := Vy^9$.
 - In fact, g 's a profunctor: it preserves the terminal, $(g \circ V) \cong V$.
- 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 $V \xrightarrow{i} C$:



From here you can iteratively “run” the cellular automaton.

Running the cellular automaton



Use that $Vy \xleftarrow{V} 0$ is terminal and $Vy \xleftarrow{g} Vy$ preserves terminals.

What is deep learning?

In *Backprop as functor*⁴ “deep learning” is expressed in terms of SMCs.

- Objects are Euclidean spaces \mathbb{R}^n ; monoidal product is \times .
- A morphism $\mathbb{R}^m \rightsquigarrow \mathbb{R}^n$ consists of
 - Another Euclidean space \mathbb{R}^p , *parameter space*,
 - A function $I: \mathbb{R}^p \times \mathbb{R}^m \rightarrow \mathbb{R}^n$, *implement*
 - A function $U: \mathbb{R}^p \times \mathbb{R}^m \times \mathbb{R}^n \rightarrow \mathbb{R}^p \times \mathbb{R}^m$, *update and backprop*
- Explanation:
 - The update takes an (inp, outp) pair and updates the parameter.
 - Without backprop, morphism composition cannot be defined.

⁴Fong, B; Spivak, DI; Tuyéras, R. “Backprop as functor”. *LICS 2019*.

What is deep learning?

In *Backprop as functor*⁴ “deep learning” is expressed in terms of SMCs.

- Objects are Euclidean spaces \mathbb{R}^n ; monoidal product is \times .
- A morphism $\mathbb{R}^m \rightsquigarrow \mathbb{R}^n$ consists of
 - Another Euclidean space \mathbb{R}^p , *parameter space*,
 - A function $I: \mathbb{R}^p \times \mathbb{R}^m \rightarrow \mathbb{R}^n$, *implement*
 - A function $U: \mathbb{R}^p \times \mathbb{R}^m \times \mathbb{R}^n \rightarrow \mathbb{R}^p \times \mathbb{R}^m$, *update and backprop*
- Explanation:
 - The update takes an (inp, outp) pair and updates the parameter.
 - Without backprop, morphism composition cannot be defined.
- Typically, I and U have very particular forms.
 - I is usu. a composite of linear maps and logistic-like maps.
 - U is usu. gradient descent along a “loss covector” $\ell \in T^*(\mathbb{R}^n) \cong \mathbb{R}^n$.

⁴Fong, B; Spivak, DI; Tuyéras, R. “Backprop as functor”. *LICS 2019*.

Deep learning in Poly

The best-known methods use calculus, but the structure is set-theoretic.

$$\mathbf{Learn}(A, B) := \{(P, I, U) \mid P \in \mathbf{Set}, I: P \times A \rightarrow B, U: P \times A \times B \rightarrow P \times A\}$$

We can see this inside of **Poly**:

$$\mathbf{Learn}(A, B) \cong [Ay^A, By^B]\text{-}\mathbf{Coalg}$$

That is, it's the cat'y of dynamical systems in $[Ay^A, By^B]$, where recall

$$[Ay^A, By^B] \cong \sum_{\varphi: Ay^A \rightarrow By^B} y^{AB}$$

An (A, B) -learner is thus a set P and a map $P \rightarrow [Ay^A, By^B] \triangleleft P$.

Learners' languages

For any polynomial p , the category $p\text{-}\mathbf{Coalg}$ forms a topos.

- Indeed, letting \mathfrak{c}_p be the cofree comonoid on p, \dots
- ...there is an equivalence $p\text{-}\mathbf{Coalg} \cong \mathfrak{c}_p\text{-}\mathbf{Set}$.
- Since \mathfrak{c}_p is free on a graph, $\mathfrak{c}_p\text{-}\mathbf{Set}$ is about as easy as toposes get.

Learners' languages

For any polynomial p , the category $p\text{-}\mathbf{Coalg}$ forms a topos.

- Indeed, letting c_p be the cofree comonoid on p ,...
- ...there is an equivalence $p\text{-}\mathbf{Coalg} \cong \mathsf{c}_p\text{-}\mathbf{Set}$.
- Since c_p is free on a graph, $\mathsf{c}_p\text{-}\mathbf{Set}$ is about as easy as toposes get.

In particular, the topos $p\text{-}\mathbf{Coalg}$ has an internal type theory and logic.

- The logic describes constraints on dynamical systems.
- A proposition ϕ is any subobject of the terminal p -coalgebra:
- a set ϕ of p -trees where if $t \in \phi$ then so is the subtree at any node.

Learners' languages

For any polynomial p , the category $p\text{-Coalg}$ forms a topos.

- Indeed, letting c_p be the cofree comonoid on p, \dots
- ...there is an equivalence $p\text{-Coalg} \cong c_p\text{-Set}$.
- Since c_p is free on a graph, $c_p\text{-Set}$ is about as easy as toposes get.

In particular, the topos $p\text{-Coalg}$ has an internal type theory and logic.

- The logic describes constraints on dynamical systems.
- A proposition ϕ is any subobject of the terminal p -coalgebra:
- a set ϕ of p -trees where if $t \in \phi$ then so is the subtree at any node.

Gradient descent-backprop is a proposition in $[\mathbb{R}^m y^{\mathbb{R}^m}, \mathbb{R}^n y^{\mathbb{R}^n}]\text{-Coalg}$.

- That is, it is a constraint on $(\mathbb{R}^m, \mathbb{R}^n)$ -learners.
- It has a very particular flavor: it can be checked in one timestep.

But the logic is much more expressive. We'll leave that for a later time.

Outline

1 Introduction

2 Theory

3 Applications

4 **Conclusion**

- Summary

Summary

Poly is a category of remarkable abundance.

- It's completely combinatorial.
 - Calculations using “the abacus” 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.
- It's got a wide scope of applications.
 - Databases and data migration.
 - Dynamical systems and cellular automata.
 - Deep learning and its generalizations.

Thank you for your time; questions and comments welcome.