The Polynomial Abacus

David I. Spivak

TOPOS INSTITUTE

FMCS 2024
2024 July 09

Outline

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

First session:

- Introduce **Poly** and its combinatorics (how the abacus works);
- Discuss its pleasing properties and monoidal structures;
- Present the framed bicategory $\mathbb{C}\mathbf{at}^\sharp$.

Second session:

- Recall $\mathbb{C}\mathbf{at}^\sharp$ and discuss some properties of it;
- Consider applications: dynamical systems, data, and deep learning;
- Conclude with a summary.

Outline

Poly for experts

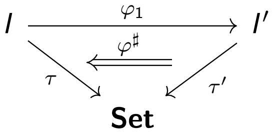
What I'll call the category **Poly** has many names.

- The free completely distributive category on one object;
- The free coproduct completion of $\mathbf{Set}^{\mathrm{op}}$;
- 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;

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 (I, τ), where $I \in \mathbf{Set}$ and $\tau \colon I \to \mathbf{Set}$.
    - Morphisms $(I, \tau) \xrightarrow{\varphi} (I', \tau')$: pairs $(\varphi_1, \varphi^\sharp)$, where



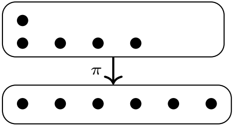
But let's make this easier.

What is a polynomial?

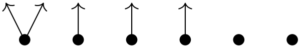
Algebraic

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

Bundle



Corolla forest

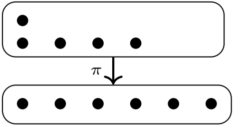


What is a polynomial?

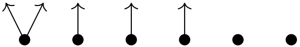
Algebraic

$y^2 + 3y + 2$

Bundle



Corolla forest



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

Terminology woes

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

$$p \coloneqq y^3 + y^2 + y^2 + 1$$

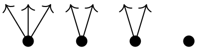
- Container terminology from Abbott: "shapes and positions".
  - `data p Y = Foo Y Y Y | Bar Y Y | Baz Y Y | Qux`
  - Container p has four "shapes", e.g. `Foo` has three "positions".

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*.

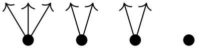


Terminology woes

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*.

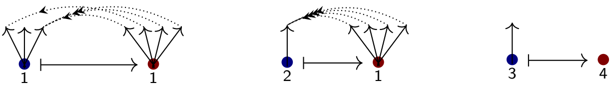


- Hard decision but I'll say positions and directions. Reasons:
  - Dynamical systems: position = point, direction = vector.
  - Categories: position = object, direction = morphism.
  - Terminal coalgebra trees: position = label, direction = arrow.

Combinatorics of polynomial morphisms

Let $p \coloneqq y^3 + 2y$ and $q \coloneqq y^4 + y^2 + 2$

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



Example: how to think of

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

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.

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

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

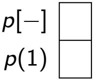
Here's a derivation of the combinatorial formula for morphisms:

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

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

Notation for the abacus

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



Notation for the abacus

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

$p[-]$	
$p(1)$	i

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

Notation for the abacus

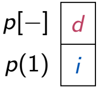
For any polynomial $p \in \mathbf{Poly}$, I'll use the following sort of notation

$$\begin{array}{r|c|} \cline{2-2} p[-] & d \\ \cline{2-2} p(1) & i \\ \cline{2-2} \end{array}$$

- 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 $d \in p[i]$.

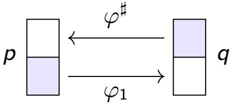
Notation for the abacus

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



- 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 $d \in p[i]$.

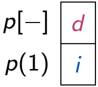
This gets more interesting for maps. A map $\varphi \colon p \to q$ is shown:



The map φ is a formula saying "however you fill blue's, I'll fill whites."

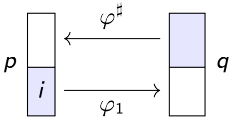
Notation for the abacus

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



- 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 $\varphi\colon p \to q$ is shown:

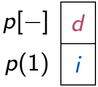


The map φ is a formula saying "however you fill blue's, I'll fill whites."

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

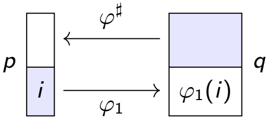
Notation for the abacus

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



- 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 $\varphi \colon p \to q$ is shown:

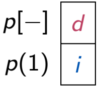


The map φ is a formula saying "however you fill blue's, I'll fill whites."

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

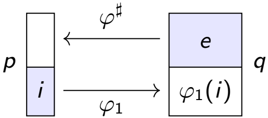
Notation for the abacus

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



- 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 $\varphi \colon p \to q$ is shown:

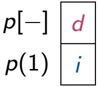


The map φ is a formula saying "however you fill blue's, I'll fill whites."

- 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,

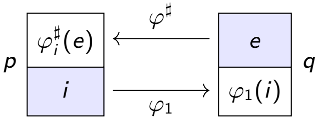
Notation for the abacus

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



- 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 $\varphi\colon p \to q$ is shown:

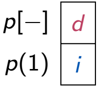


The map φ is a formula saying "however you fill blue's, I'll fill whites."

- 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 $\varphi^\sharp_i(e) \in p[i]$.

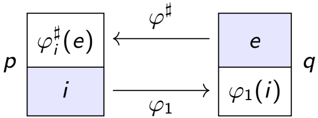
Notation for the abacus

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



- 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 $\varphi\colon p \to q$ 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 $\varphi_i^\sharp(e) \in p[i]$.

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 $\mathbf{Set}^{\mathsf{op}}$.
  - 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 $\mathbf{Set}^{\mathrm{op}}$.
  - Sets A can be represented as a constant or linear: $A, Ay \in$ **Poly**.
  - Sets A can be op-represented as representables $y^A \in$ **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 " $\mathbf{Set}^{\mathbf{Set}}$ ".
  - 0 is initial, 1 is terminal, $+$ is coproduct, $\times$ is product.
  - y^A is internal hom between $A, y \in$ **Poly**. For fun: $y^y \cong y + 1$.
- **Poly** has coequalizers, though these differ from coeq's in " $\mathbf{Set}^{\mathbf{Set}}$ ".

Pleasing aspects of Poly

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

- **Poly** contains two copies of **Set** and one copy of $\mathbf{Set}^{\text{op}}$.
  - Sets A can be represented as a constant or linear: $A, Ay \in$ **Poly**.
  - Sets A can be op-represented as representables $y^A \in$ **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 " $\mathbf{Set}^{\mathbf{Set}}$ ".
  - 0 is initial, 1 is terminal, $+$ is coproduct, $\times$ is product.
  - y^A is internal hom between $A, y \in$ **Poly**. For fun: $y^y \cong y + 1$.
- **Poly** has coequalizers, though these differ from coeq's in " $\mathbf{Set}^{\mathbf{Set}}$ ".
- **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 $(y, \otimes)$.

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

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 $(y, \otimes)$.

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

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

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

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 $(y, \otimes)$.

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

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

$$[p, q] \coloneqq \sum_{\varphi \colon p \to q} y^{\sum_{i \in p(1)} q[\varphi_1(i)]}$$

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

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

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: $p \triangleleft y \cong p \cong y \triangleleft p$.

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: $p \triangleleft y \cong p \cong y \triangleleft p$.

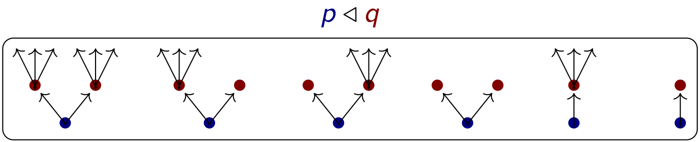
Why the we weird symbol $\triangleleft$ rather than $\circ$?

- 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 $p := y^2 + y$ 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.

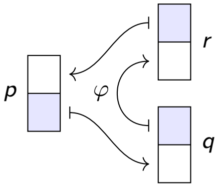
Maps to composites

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

- A map $\varphi \colon p \to q \triangleleft r$ 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:



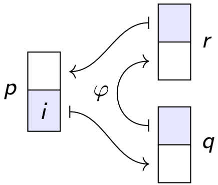
Maps to composites

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

- A map $\varphi\colon p \to q \triangleleft r$ 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:



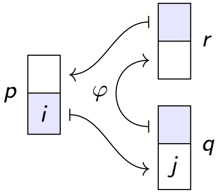
Maps to composites

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

- A map $\varphi\colon p \to q \triangleleft r$ 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:



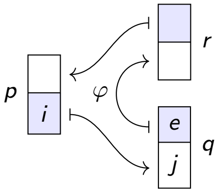
Maps to composites

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

- A map $\varphi \colon p \to q \triangleleft r$ 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:



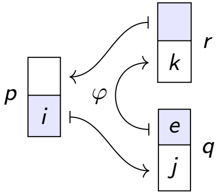
Maps to composites

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

- A map $\varphi\colon p \to q \triangleleft r$ 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:



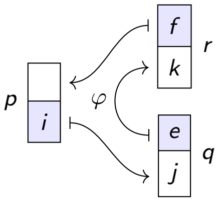
Maps to composites

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

- A map $\varphi\colon p \to q \triangleleft r$ 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:



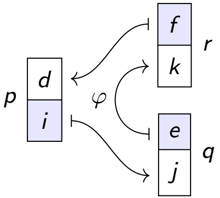
Maps to composites

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

- A map $\varphi\colon p \to q \triangleleft r$ 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 $(\textbf{Poly}, y, \triangleleft)$

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]

[1] Ahman-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*,
  - $\epsilon\colon m \to I$ 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]

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

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

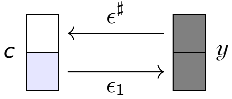
  where $\mathcal{C}[i]$ is the set of morphisms in $\mathcal{C}$ that emanate from i.
- The counit $\epsilon\colon \mathfrak{c} \to y$ assigns to each object an identity.
- The comult $\delta\colon \mathfrak{c} \to \mathfrak{c} \triangleleft \mathfrak{c}$ assigns codomains and composites.

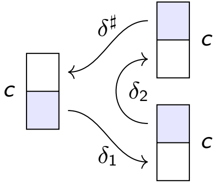
[1] Ahman-Uustalu. "Directed Containers as Categories". *MSFP 2016*.

The abacus in action

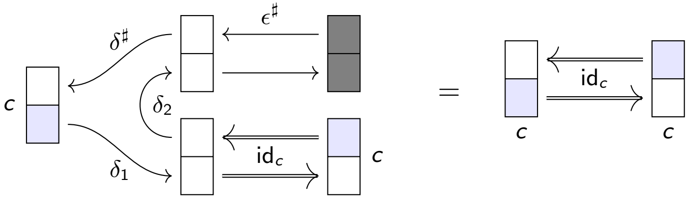
We can understand the Ahman-Uustalu result combinatorially.

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





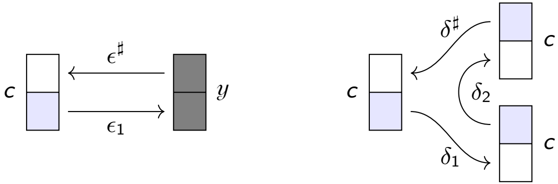
Here's the first unitality law, $(\mathrm{id}_c \triangleleft \epsilon) \circ \delta = \mathrm{id}_c$:



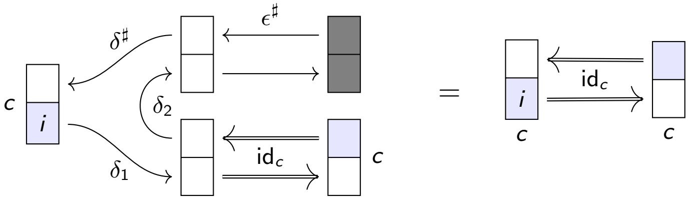
The abacus in action

We can understand the Ahman-Uustalu result combinatorially.

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



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

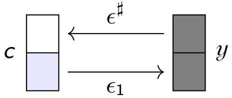


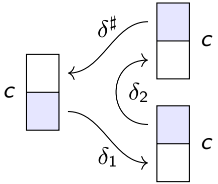
Equation: $\forall i \in c(1)$....

The abacus in action

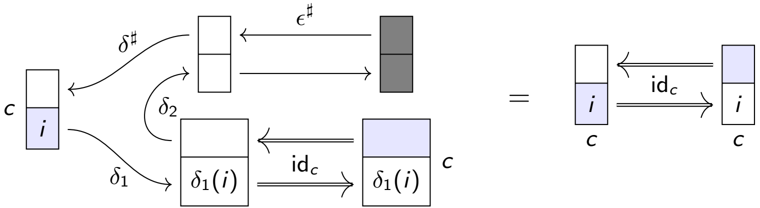
We can understand the Ahman-Uustalu result combinatorially.

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





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

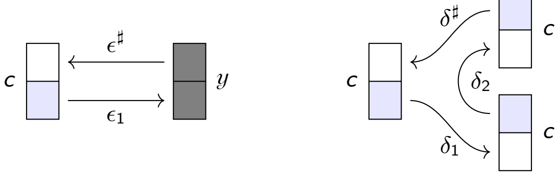


Equation: $\forall i \in c(1), \delta_1(i) = i \wedge \ldots$

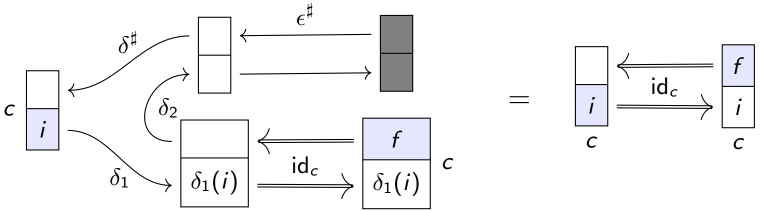
The abacus in action

We can understand the Ahman-Uustalu result combinatorially.

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



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

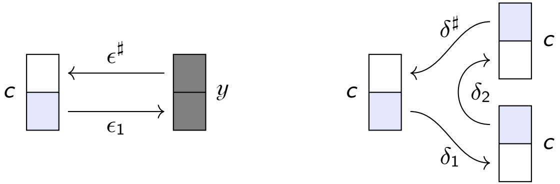


Equation: $\forall i \in c(1), \delta_1(i) = i \wedge \forall f \in c[i], \ldots$

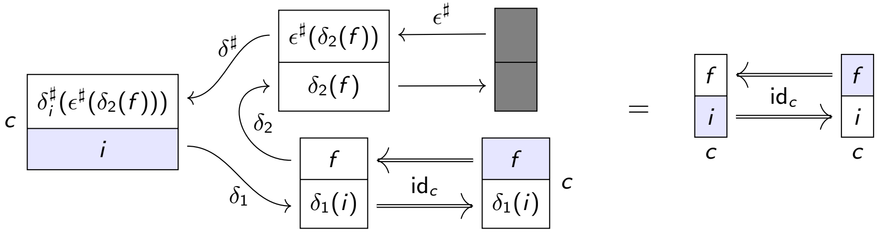
The abacus in action

We can understand the Ahman-Uustalu result combinatorially.

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



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

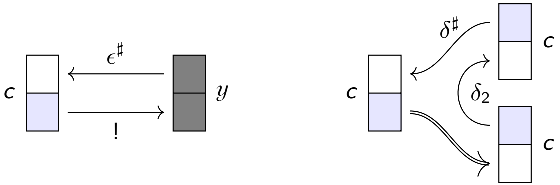


Equation: $\forall i \in c(1), \delta_1(i) = i \wedge \forall f \in c[i], \delta^\sharp_i(f, \epsilon^\sharp(\delta_2(f))) = f.$

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...

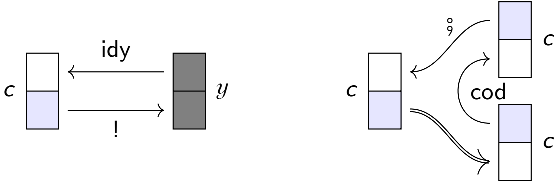


- To make sense of the other equations, let's rename $\epsilon^\sharp, \delta_2$, and $\delta^\sharp$.

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...

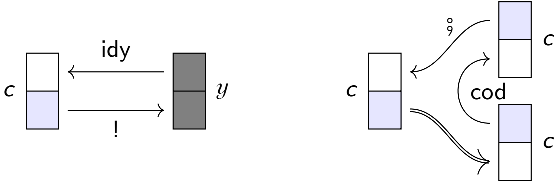


- To make sense of the other equations, let's rename $\epsilon^\sharp, \delta_2$, and $\delta^\sharp$.
- Namely, let's write idy $:= \epsilon^\sharp$, cod $:= \delta_2$, and $⨟ := \delta^\sharp$.
  - Then the previous equation says: $f ⨟ \text{idy}(\text{cod}(f)) = f$.

Making sense of the results

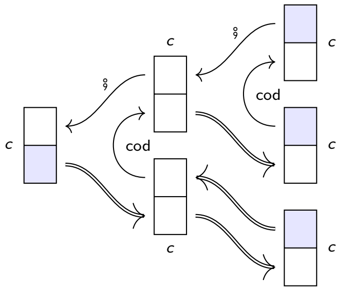
We want to make sense of the set-theoretic equations from the abacus.

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



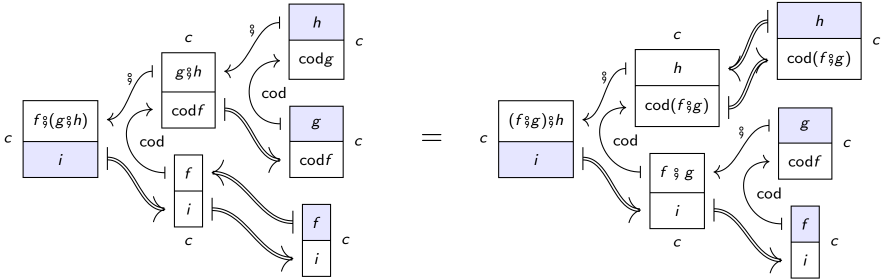
- To make sense of the other equations, let's rename $\epsilon^\sharp, \delta_2$, and $\delta^\sharp$.
- Namely, let's write idy $:= \epsilon^\sharp$, cod $:= \delta_2$, and $⨟ := \delta^\sharp$.
  - Then the previous equation says: $f ⨟ \text{idy}(\text{cod}(f)) = f$.
  - The other unitality eq'n gives: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

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:

$$\forall i \in c(1), i = i \wedge$$
$$\forall f \in c[i], \mathrm{cod} f = \mathrm{cod} f \wedge$$
$$\forall g \in c[\mathrm{cod} f], \mathrm{cod} g = \mathrm{cod}(f \mathbin{⨟} g) \wedge$$
$$\forall h \in c[\mathrm{cod} g], f \mathbin{⨟} (g \mathbin{⨟} h) = (f \mathbin{⨟} g) \mathbin{⨟} h.$$

Comonoid maps are "retrofunctors"

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

- A comonoid morphism $\varphi \colon \mathcal{C} \nrightarrow \mathcal{D}$ is called a *retrofunctor*.
- 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: $\varphi^\sharp$ preserves ids and comps, and φ_1 preserves cods.
- Denote this by $\mathbf{Cat}^\sharp := \mathbf{Comon}(\mathbf{Poly}) =$ (cat'ys and retrofunctors).

Comonoid maps are "retrofunctors"

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

- A comonoid morphism $\varphi\colon \mathcal{C} \nrightarrow \mathcal{D}$ is called a *retrofunctor*.
- 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 $\varphi^\sharp_i(f) \in \mathfrak{c}[i]$.
  - Rules: $\varphi^\sharp$ preserves ids and comps, and φ_1 preserves cods.
- Denote this by $\mathbf{Cat}^\sharp := \mathbf{Comon}(\mathbf{Poly}) =$ (cat'ys and retrofunctors).

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

- It is trivial on objects $i \in \mathrm{Ob}(\mathcal{C})$. Passing back morphisms gives:
- ... a map $\varphi^\sharp_i(q)\colon i \to i_{+q}$ emanating from i for each $q \in \mathbb{Q}$, s.t....
- ... $\varphi^\sharp_i(0) = \mathrm{id}_i$, so $i_{+0} = i$, and $\varphi^\sharp_i(q) \mathbin{⨟} \varphi^\sharp_{i_{+q}}(q') = \varphi^\sharp_i(q + q')$.

Comonoid maps are "retrofunctors"

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

- A comonoid morphism $\varphi\colon \mathcal{C} \nrightarrow \mathcal{D}$ is called a *retrofunctor*.
- 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: $\varphi^\sharp$ preserves ids and comps, and φ_1 preserves cods.
- Denote this by $\mathbf{Cat}^\sharp := \mathbf{Comon}(\mathbf{Poly}) =$ (cat'ys and retrofunctors).

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

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

"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.

$\mathbf{Cat}^\sharp$: 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.
- The following are equivalent:
  - the comonoid structure maps ϵ, δ are cartesian;
  - $\mathfrak{c} = Oy$ is a linear polynomial;
  - $\mathcal{C}$ is a discrete category, with $\mathrm{Ob}(\mathcal{C}) = O$.
- $\mathfrak{c} = y^M$ is representable iff $M \in \mathbf{Set}$ carries a monoid.
- If $\mathcal{C} = \boxed{\overset{1}{\bullet} \to \overset{2}{\bullet} \to \cdots \to \overset{N}{\bullet}}$ then $\mathfrak{c} = y^N + y^{N-1} + \cdots + y$.

$\mathbf{Cat}^\sharp$: 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.
- The following are equivalent:
  - the comonoid structure maps ϵ, δ are cartesian;
  - $\mathfrak{c} = Oy$ is a linear polynomial;
  - $\mathcal{C}$ is a discrete category, with $\mathrm{Ob}(\mathcal{C}) = O$.
- $\mathfrak{c} = y^M$ is representable iff $M \in \mathbf{Set}$ carries a monoid.
- If $\mathcal{C} = \boxed{\overset{1}{\bullet} \to \overset{2}{\bullet} \to \cdots \to \overset{N}{\bullet}}$ then $\mathfrak{c} = y^N + y^{N-1} + \cdots + y$.

Other facts about $\mathbf{Cat}^\sharp$:

- Coproducts in $\mathbf{Cat}^\sharp$ and in $\mathbf{Cat}$ agree; carrier is $\mathfrak{c} + \mathfrak{d}$.
- $\mathbf{Cat}^\sharp$ has finite products (Niu), and they're very interesting.
- $\mathbf{Cat}^\sharp$ inherits $\otimes$ from $\mathbf{Poly}$, 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 $\mathbf{Cat}^\sharp \to \mathbf{Poly}$ 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 $\mathfrak{c}_p \to y$ and $\mathfrak{c}_p \to \mathfrak{c}_p \triangleleft \mathfrak{c}_p$.

Cofree comonoids

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

- That is, the forgetful functor $\mathbf{Cat}^\sharp \to \mathbf{Poly}$ 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 $\mathfrak{c}_p \to y$ and $\mathfrak{c}_p \to \mathfrak{c}_p \triangleleft \mathfrak{c}_p$.

- 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 $\mathfrak{c}_p$ by taking the limit of the following diagram in **Poly**:

$$\mathfrak{c}_p := \lim \left(y \longleftarrow py \leftleftarrows py \triangleleft py \Lleftarrow py \triangleleft py \triangleleft py \Lleftarrow \cdots \right)$$

Cofree comonoids

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

- That is, the forgetful functor $\mathbf{Cat}^\sharp \to \mathbf{Poly}$ 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 $\mathfrak{c}_p \to y$ and $\mathfrak{c}_p \to \mathfrak{c}_p \triangleleft \mathfrak{c}_p$.

- 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 $\mathfrak{c}_p$ by taking the limit of the following diagram in **Poly**:

$$\mathfrak{c}_p := \lim \left(y \longleftarrow py \Longleftarrow py \triangleleft py \Lleftarrow py \triangleleft py \triangleleft py \Lleftarrow \cdots \right)$$

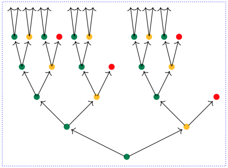
For us, a main use of $\mathfrak{c}_p$ is an equivalence $\mathfrak{c}_p$-**Set** $\cong$ p-**Coalg**.

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

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

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

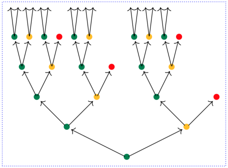
- It's actually free on a graph, but the graph is very interesting.
- The vertex-set $\mathfrak{c}_p(1)$ of the graph is the set of p-trees.
  - A p-tree is a possibly infinite tree t, where each node...
  - ...is labeled by a position $i \in p(1)$ and has $p[i]$-many branches.
  - Example object $t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



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

Comonoids in **Poly** are categories, so $\mathfrak{c}_p$ 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 $p = \{\color{orange}\bullet, \color{green}\bullet\}y^2 + \{\color{red}\bullet\} \cong 2y^2 + 1$:



- 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 $\mathfrak{c}_p$: every possible "destiny" is included.

Bicomodules in (**Poly**$, y, \triangleleft$)

Given ~~comonoids~~ categories $\mathscr{C}, \mathscr{D}$, a $(\mathscr{C}, \mathscr{D})$-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 $(\mathbf{Poly}, y, \triangleleft)$

Given ~~comonoids~~ categories $\mathcal{C}, \mathcal{D}$, 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 $\mathfrak{c}, \mathfrak{d}$.
- I denote this $(\mathcal{C}, \mathcal{D})$-bicomodule *m* like so:

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

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

$$\mathcal{C} \triangleleft\!\xrightarrow{\;m\;}\!\triangleleft\; \mathcal{D} \qquad \text{or} \qquad \mathfrak{c} \triangleleft\!\xrightarrow{\;m\;}\!\triangleleft\; \mathfrak{d}$$

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

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

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.
- Assuming Garner's result, 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

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

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.
- Assuming Garner's result, check: ${}_{\mathcal{C}}\mathbf{Mod}_0 \cong \mathcal{C}\text{-}\mathbf{Set}$.

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

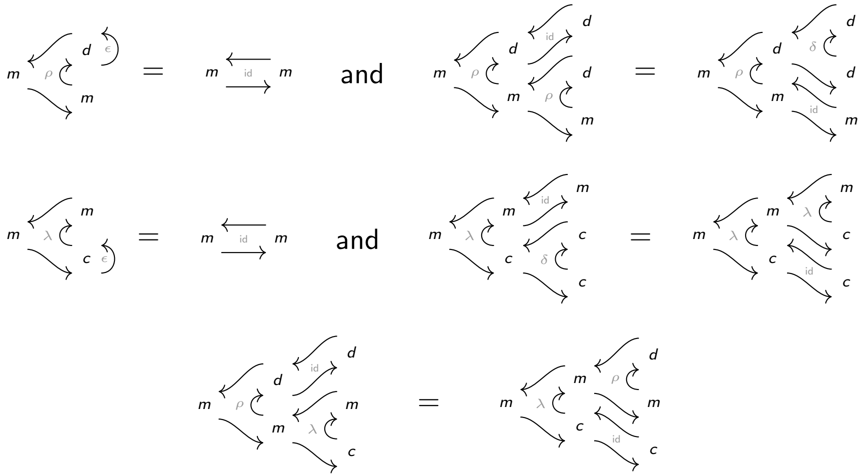
- A profunctor $\mathcal{C} \nrightarrow \mathcal{D}$ is a functor $\mathcal{C} \to (\mathcal{D}\text{-}\mathbf{Set})^{\mathrm{op}}$
- A prafunctor $\mathcal{C} \triangleleft\!\longrightarrow\!\triangleleft \mathcal{D}$ is a functor $\mathcal{C} \to \mathbf{Coco}\big((\mathcal{D}\text{-}\mathbf{Set})^{\mathrm{op}}\big)$...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

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

Let's ask the abacus

To prove that bicomodules $\mathfrak{c} \overset{m}{\triangleleft\!\!-\!\!\!-\!\!\triangleleft} \mathfrak{d}$ are prafunctors ${}_{\mathfrak{d}}\mathbf{Mod}_0 \to {}_{\mathfrak{c}}\mathbf{Mod}_0$:

- Write out the bicomodule equations and run the abacus.



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 $\mathfrak{c} \to$ **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 $m[x]$ ...
- ... 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 $\mathfrak{c} \to$ **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 $m[x]$ ...
- ... 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 $\mathfrak{d}$-**Set** $\to$ $\mathfrak{c}$-**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 $\mathfrak{c} \overset{m}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} \mathfrak{d}$.

- 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 $\mathfrak{c} \overset{M}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} 0 \overset{!}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} \mathfrak{d}$.

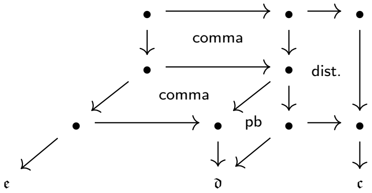
Getting acquainted with bicomodules

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

- 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 $\mathfrak{c} \triangleleft\!\xrightarrow{M}\!\triangleleft\ 0 \triangleleft\!\xrightarrow{!}\!\triangleleft\ \mathfrak{d}$.
- The following cat'ies are isomorphic and all are equivalent to $\mathfrak{c}$-**Set**:
    - Cartesian retrofunctors 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 $\mathfrak{c} \xleftarrow{m} \mathfrak{d} \xleftarrow{n} \mathfrak{e}$, is carried by the equalizer:

$$m \triangleleft_{\mathfrak{d}} n \xrightarrow{eq} m \triangleleft n \rightrightarrows m \triangleleft \mathfrak{d} \triangleleft n$$

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

The framed bicategory $\mathbb{C}\mathrm{at}^\sharp$

Poly como'ds, retrofuns, and bicomodules form a framed bicategory $\mathbb{C}\mathbf{at}^\sharp$.

$$\begin{array}{ccc} \mathfrak{c} & \overset{m}{\lhd\!\!\longleftarrow\!\!\lhd} & \mathfrak{d} \\ {\scriptstyle\varphi}\big\downarrow & \Downarrow\alpha & \big\downarrow{\scriptstyle\psi} \\ \mathfrak{c}' & \underset{m'}{\lhd\!\!\longleftarrow\!\!\lhd} & \mathfrak{d}' \end{array}$$

- 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 ${}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}}$ for categories $\mathcal{C}, \mathcal{D}$.

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

The framed bicategory $\mathbb{C}\mathrm{at}^\sharp$

Poly como'ds, retrofuns, and bicomodules form a framed bicategory $\mathbb{C}\mathbf{at}^\sharp$.

$$\begin{array}{ccc} \mathfrak{c} & \overset{m}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} & \mathfrak{d} \\ {\scriptstyle\varphi}\big\downarrow & \Downarrow\alpha & \big\downarrow{\scriptstyle\psi} \\ \mathfrak{c}' & \underset{m'}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} & \mathfrak{d}' \end{array}$$

- 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 ${}_{\mathcal{C}}\mathbf{Mod}_{\mathcal{D}}$ for categories $\mathcal{C}, \mathcal{D}$.

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

There's a factorization system on $\mathbb{C}\mathbf{at}^\sharp$:

- Every $m \in {}_{\mathfrak{c}}\mathbf{Mod}_{\mathfrak{d}}$ can be factored as $m \cong f \circ p$,

$$\mathfrak{c} \overset{f}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} \mathfrak{c}' \overset{p}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} \mathfrak{d}$$

where f "is" a discrete opfibration and p "is" a profunctor.

Gambino-Kock's framed bicategory $\mathbb{P}\mathrm{oly}$

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

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

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

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

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

This is a full subcategory $\mathbb{P}\mathbf{oly} \subseteq \mathbb{C}\mathbf{at}^\sharp$.

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

The comonoid theory $\mathbb{C}\mathbf{at}^\sharp$ of (one-variable) **Poly** includes all of $\mathbb{P}\mathbf{oly}$.

Adjunctions in $\mathbb{C}\mathbf{at}^\sharp$

The map $\_\mathbf{Mod}_0\colon (\mathbb{C}\mathbf{at}^\sharp)^{\mathrm{op}} \to \mathbb{C}\mathbf{at}$ is locally fully faithful; i.e....

- ...for categories $\mathcal{C}, \mathcal{D}$, only some functors $m\colon \mathcal{D}\text{-}\mathbf{Set} \to \mathcal{C}\text{-}\mathbf{Set}$ count...
- ... as bimodules $\mathcal{C} \overset{m}{\triangleleft\!\!-\!\!\!-\!\!\triangleleft} \mathcal{D}$, 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 $\mathcal{C} \overset{m}{\triangleleft\!\!-\!\!\!-\!\!\triangleleft} \mathcal{D}$ has an adjoint in $\mathbb{C}\mathbf{at}^\sharp$, 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 $\mathbb{C}\mathbf{at}^\sharp$! (i.e. the adjoint m' needs to preserve conn'd limits).

Adjunctions in $\mathbb{C}\mathbf{at}^\sharp$

The map $\_\mathbf{Mod}_0 \colon (\mathbb{C}\mathbf{at}^\sharp)^{\mathrm{op}} \to \mathbb{C}\mathbf{at}$ is locally fully faithful; i.e....

- ...for categories $\mathcal{C}, \mathcal{D}$, only some functors $m \colon \mathcal{D}\text{-}\mathbf{Set} \to \mathcal{C}\text{-}\mathbf{Set}$ count...
- ... as bimodules $\mathcal{C} \overset{m}{\lhd\!\!\longrightarrow\!\!\lhd} \mathcal{D}$, 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 $\mathcal{C} \overset{m}{\lhd\!\!\longrightarrow\!\!\lhd} \mathcal{D}$ has an adjoint in $\mathbb{C}\mathbf{at}^\sharp$, 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 $\mathbb{C}\mathbf{at}^\sharp$! (i.e. the adjoint m' needs to preserve conn'd limits).

Both functors $\mathcal{C} \xrightarrow{F} \mathcal{D}$ and retrofun's $\mathcal{C} \overset{\varphi}{\nrightarrow} \mathcal{D}$ induce adjunctions in $\mathbb{C}\mathbf{at}^\sharp$.

- 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 retrofunctor, and the Δ's coincide.

Note that retrofunctors $\mathcal{C} \nrightarrow \mathcal{D}$ induce interesting maps between toposes:

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

Operads as monads in $\mathbb{C}\mathrm{at}^\sharp$

In any framed bicat'y, notation from $\mathbb{C}\mathbf{at}^\sharp$, a monad $(\mathcal{C}, m, \eta, \mu)$ consists of

- An object $\mathcal{C}$, the *type*
- a bicomodule $\mathcal{C} \triangleleft\!\xrightarrow{m}\!\triangleleft\ \mathcal{C}$, 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.

Operads as monads in $\mathbb{C}\mathbf{at}^\sharp$

In any framed bicat'y, notation from $\mathbb{C}\mathbf{at}^\sharp$, a monad $(\mathcal{C}, m, \eta, \mu)$ consists of

- An object $\mathcal{C}$, the *type*
- a bicomodule $\mathcal{C} \triangleleft\!\xrightarrow{m}\!\triangleleft\ \mathcal{C}$, 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 $\mathbb{C}\mathbf{at}^\sharp$, 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.

[3] Not 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 $\mathbb{C}\mathbf{at}^\sharp$ setting; see Gambino-Kock.

"Categories = monads in $\mathbb{S}$pan" in $\mathbb{C}$at$^\sharp$

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

- A prafunctor $Oy \overset{m}{\rhd\!\!\longrightarrow\!\!\rhd} Oy$ acts as a span iff it's a left adjoint.
- If a monad m has a right adjoint $Oy \overset{c}{\lhd\!\!\longleftarrow\!\!\lhd} Oy$, then c is a comonad.
- Now, since the vertical part of $\mathbb{C}\mathbf{at}^\sharp$ is already **Comon**(**Poly**),
- ... c has a canonical comonoid structure $\mathfrak{c}$, equipped with $\mathfrak{c} \nrightarrow Oy$.
- This map $\mathfrak{c} \nrightarrow Oy$ 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 $\mathbb{S}$pan" in $\mathbb{C}$at$^\sharp$

It is well-known that "categories are monads in $\mathbb{S}$**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 $\mathbb{C}\mathbf{at}^\sharp$ is already **Comon**(**Poly**),
- … c has a canonical comonoid structure $\mathfrak{c}$, equipped with $\mathfrak{c} \nrightarrow Oy$.
- This map $\mathfrak{c} \nrightarrow Oy$ 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 retrofunctors look in this perspective:

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

Grothendieck sites give $\mathbb{C}\mathbf{at}^\sharp$-monads

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

- 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 $\mathbb{C}\mathbf{at}^\sharp$-monads

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

- 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 $\mathcal{C} \triangleleft\!\xrightarrow{m}\!\triangleleft\ \mathcal{C}$.

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

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

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

Grothendieck sites give $\mathbb{C}\mathbf{at}^\sharp$-monads

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

- 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.

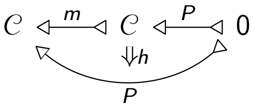
To each Grothendieck top'y J, we need (m, η, μ) where $\mathcal{C} \overset{m}{\triangleleft\!\!-\!\!\triangleleft} \mathcal{C}$.

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

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

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

Bringing the abacus out of the monastery

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

- **Poly** and $\mathbb{C}\mathbf{at}^\sharp$ 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 $r\colon S \to B$, 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* 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* of the Moore machine.

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

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

Moore machines as maps in Poly

We can understand Moore machines 

in terms of polynomials.

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

Moore machines as maps in Poly

We can understand Moore machines 

in terms of polynomials.

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

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

- For arbitrary $p \in$ **Poly** we can interpret a map $\varphi: S \to p \triangleleft S$ 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 

in terms of polynomials.

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

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

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

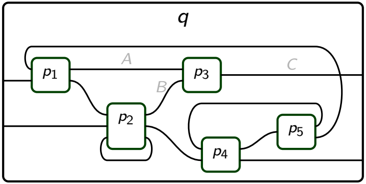
Even more general: a functor $S\colon \mathcal{C} \to \mathbf{Set}$ for any category $\mathcal{C}$.

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



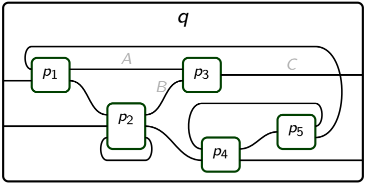
(φ)

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 $\varphi\colon p_1 \otimes \cdots \otimes p_5 \to q$.
  - 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 $\varphi\colon p_1 \otimes \cdots \otimes p_5 \to q$.
  - 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 $S_i \to p_i \triangleleft S_i$.
- We tensor and compose to give $S \to q \triangleleft S$, where $S \coloneqq S_1 \times \cdots \times S_5$.

So φ applied to dynamics in $p_1, \ldots, p_5$ 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 $Wy \otimes Wy \otimes 2y^W \to y$.

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

Other sorts of dynamical systems

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

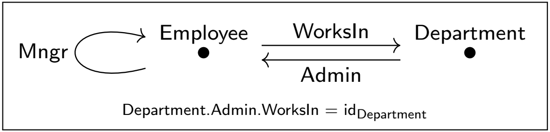
- Discrete: $\mathbb{N}$, reversible: $\mathbb{Z}$, real-time: $\mathbb{R}$.
- If T is a monoid and S is a set, a T-action on S is equivalently...
- ... a functor $S\colon T \to \mathbf{Set}$, as in our general definition above.

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

Categorical databases

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

$\mathcal{C} :=$



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

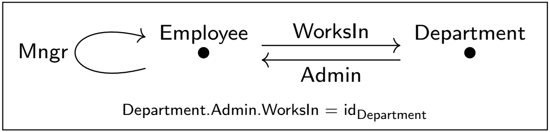
Employee	**WorksIn**	**Mngr**
♡	P9	♡
T****	bLue	orca
orca	bLue	orca

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. $\mathcal{C} \triangleleft \xrightarrow{I} \triangleleft\ 0$) can be represented as follows:

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

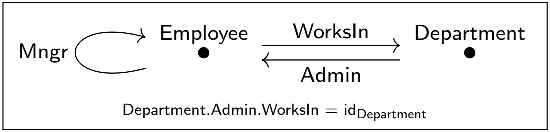
Department	**Admin**
bLue	T****
P9	♡

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:

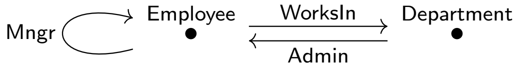
Employee	FName	WorksIn	Mngr
♡	Alan	P9	♡
T****	Dani	bLue	orca
orca	Sara	bLue	orca

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

Categorical databases

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

$\mathcal{C} :=$



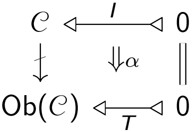
$$\text{Department.Admin.WorksIn} = \text{id}_{\text{Department}}$$

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



Data migration

The framed bicategory structure of $\mathbb{C}\mathbf{at}^\sharp$ is very useful in databases.

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

Data migration

The framed bicategory structure of $\mathbb{C}\mathbf{at}^\sharp$ is very useful in databases.

- We hinted at this in the last slide, adding attributes via a retrofunctor.
- 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 an object in ${}_{\mathbf{1}}\mathbf{Mod}_{\mathcal{D}}$?
- 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 $\mathbb{C}\mathbf{at}^\sharp$ is very useful in databases.

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

A prafunctor $\mathcal{C} \overset{P}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} \mathcal{D}$ 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 an object in ${}_{\mathbf{1}}\mathbf{Mod}_{\mathcal{D}}$?
- 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 $\mathbf{1} \overset{Q}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} \mathcal{D}$ 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...

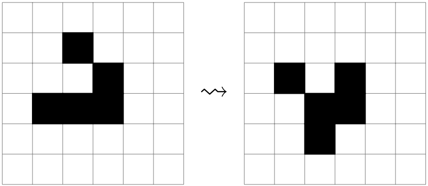
$$(\text{City} \times_{\text{State}} \text{City}) + (\text{City} \times_{\text{State}} \text{County}) + (\text{County} \times_{\text{State}} \text{County})$$

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

Cellular automata

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

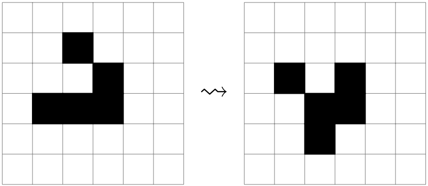
- Here's a picture of a *glider* from Conway's Game of Life:



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:

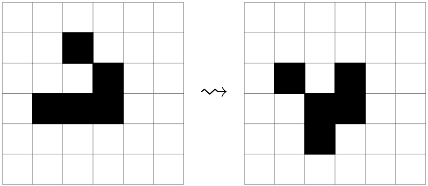


- GoL takes place on a grid, a set $V := \mathbb{Z} \times \mathbb{Z}$ of "squares"
- Each square has neighbors; think of the grid as a graph $A \rightrightarrows V$.
- Each square can be in one of two states: white or black.

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 $A \rightrightarrows V$.
- 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 $\mathbb{C}\mathbf{at}^\sharp$

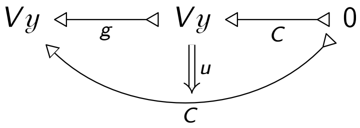
How do we encode this in $\mathbb{C}\mathbf{at}^\sharp$?

- 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 $g := Vy^9$.
    - In fact, g's a profunctor: it preserves the terminal, $(g \circ V) \cong V$.

Cellular automata as algebras in $\mathbb{C}\mathbf{at}^\sharp$

How do we encode this in $\mathbb{C}\mathbf{at}^\sharp$?

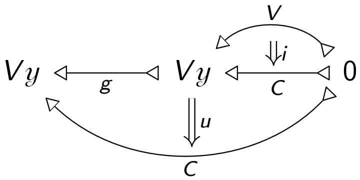
- We encode the graph $A \rightrightarrows V$ as a prafunctor $Vy \mathrel{\lhd\!\!\xleftarrow{\;g\;}\!\!\lhd} Vy$
  - Each $v \in V$ queries its neighbors (and itself).
  - The carrier of the prafunctor for GoL is $g := Vy^9$.
  - In fact, g's a profunctor: it preserves the terminal, $(g \circ V) \cong V$.
- We encode the color-set for each node as a prafunctor $Vy \mathrel{\lhd\!\!\xleftarrow{\;C\;}\!\!\lhd} 0$
  - 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 $\mathbb{C}\mathbf{at}^\sharp$

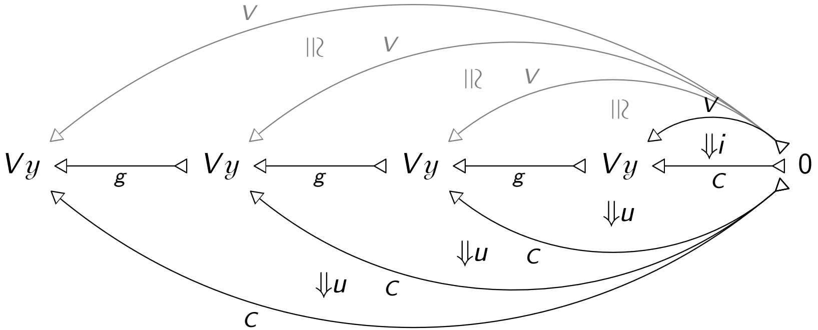
How do we encode this in $\mathbb{C}\mathbf{at}^\sharp$?

- 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 $g \coloneqq Vy^9$.
  - 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. $C \coloneqq 2V$.
- 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 $I \colon \mathbb{R}^p \times \mathbb{R}^m \to \mathbb{R}^n$, *implement*
  - A function $U \colon \mathbb{R}^p \times \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^p \times \mathbb{R}^m$, *update and backprop*
- Explanation:
  - The update takes an (inp, outp) pair and updates the parameter.
  - Without backprop, morphism composition cannot be defined.

[4] Fong, 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 $I\colon \mathbb{R}^p \times \mathbb{R}^m \to \mathbb{R}^n$, *implement*
  - A function $U\colon \mathbb{R}^p \times \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^p \times \mathbb{R}^m$, *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" $\ell \in T^*(\mathbb{R}^n) \cong \mathbb{R}^n$.

[4] Fong, 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.

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

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

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

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

Learners' languages

For any polynomial p, the category p-**Coalg** forms a topos.

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

Learners' languages

For any polynomial p, the category p-**Coalg** forms a topos.

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

In particular, the topos p-**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-**Coalg** forms a topos.

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

In particular, the topos p-**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 $[\mathbb{R}^m y^{\mathbb{R}^m}, \mathbb{R}^n y^{\mathbb{R}^n}]$-**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

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.