Functorial Aggregation

David I. Spivak

TOPOS INSTITUTE

Workshop on Polynomial Functors
2022 March 18

Outline

Why think about databases?

I'm interested in sense-making. How do we make sense of the world?

- We're here together, each with our own purpose and abilities.
- We're engaged in the activity of collective sense-making.
- I'm want to spread a *sense* of how **Poly** relates to information.

Why think about databases?

I'm interested in sense-making. How do we make sense of the world?

- We're here together, each with our own purpose and abilities.
- We're engaged in the activity of collective sense-making.
- I'm want to spread a *sense* of how **Poly** relates to information.

Imagine that sense is "contained" somewhere and that it can be transferred.

- If our ability to deal effectively with the world were contained...
- ...in our brain, then we could ask "what's the brain's data structure?"
- And if our data structures are different, then how is info transferred?
- Are you getting this? If so, what's the story of *how that works*?

Why think about databases?

I'm interested in sense-making. How do we make sense of the world?

- We're here together, each with our own purpose and abilities.
- We're engaged in the activity of collective sense-making.
- I'm want to spread a *sense* of how **Poly** relates to information.

Imagine that sense is "contained" somewhere and that it can be transferred.

- If our ability to deal effectively with the world were contained...
- ...in our brain, then we could ask "what's the brain's data structure?"
- And if our data structures are different, then how is info transferred?
- Are you getting this? If so, what's the story of *how that works*?

I think of mathematical fields as accounting systems.

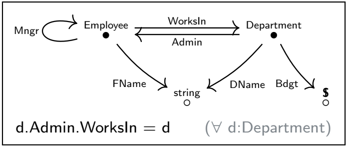
- Arithmetic accounts for the flow of quantities, as in finance.
- Hilbert spaces account for the states of elementary particles, as in QM.
- Probability distributions account for likelihoods, as in game theory.
- What's a good accounting system for how we collectively make sense?

Categorical databases

When I first started out on this question, I began with databases.

- Their mundane and humble but widely-used and easily conceptualized.
- The way I conceptualized them was as copresheaves $F\colon \mathcal{C} \to \mathbf{Set}$.
- The site $\mathcal{C}$ is called the *schema* and the copresheaf is the *instance*.

mySchema :=



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

String
Alan
IT
⋮

Department	DName	Admin	Bdgt
101	Sales	1	\$10
102	IT	3	\$5

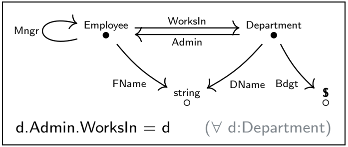
x : \$
\$5
\$6
⋮

Categorical databases

When I first started out on this question, I began with databases.

- Their mundane and humble but widely-used and easily conceptualized.
- The way I conceptualized them was as copresheaves $F \colon \mathcal{C} \to \mathbf{Set}$.
- The site $\mathcal{C}$ is called the *schema* and the copresheaf is the *instance*.

mySchema $:=$



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

String
Alan
IT
⋮

Department	DName	Admin	Bdgt
101	Sales	1	$10
102	IT	3	$5

x : $
$5
$6
⋮

- The schema holds the *structure* of your knowledge...
- ...and the instance holds all your *examples* within that structure.

For those who don't care about databases, this talk is about copresheaves.

Querying and aggregating

The two most common thing to do with databases is *query and aggregate*.

- Querying a database means performing a limit operation.
  - Example: find all pairs of people with the same favorite book.

$$\begin{tikzcd} Q \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & P \arrow[d, "f"] \\ P \arrow[r, "f"'] & B \end{tikzcd}$$

Querying and aggregating

The two most common thing to do with databases is *query and aggregate*.

- Querying a database means performing a limit operation.
  - Example: find all pairs of people with the same favorite book.

$$\begin{tikzcd} Q \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & P \arrow[d, "f"] \\ P \arrow[r, "f"'] & B \end{tikzcd}$$

  - This is called a *conjunctive query*, "an X and a Y where..."
  - More generally a *disjoint union of conj'ive queries (duc-query)*...
  - ... is a coproduct of these, e.g. same favorite book or movie.

Querying and aggregating

The two most common thing to do with databases is *query and aggregate*.

- Querying a database means performing a limit operation.
  - Example: find all pairs of people with the same favorite book.

$$\begin{tikzcd} Q \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & P \arrow[d, "f"] \\ P \arrow[r, "f"'] & B \end{tikzcd}$$

  - This is called a *conjunctive query*, "an X and a Y where..."
  - More generally a *disjoint union of conj'ive queries (duc-query)*...
  - ... is a coproduct of these, e.g. same favorite book or movie.
- Aggregating is "integrating along compact fibers".
  - Assign to each $b \in B$ the number of people whose fave book is b.

$$\begin{tikzcd} P \arrow[r, "s"] \arrow[d, "f"'] & \mathbb{R} \\ B \arrow[ur, dashed, "(\mathrm{sum}\ s)_f"'] & \end{tikzcd}$$

  - Or assign to b the total salary of everyone whose fave is b.
  - Rather than searching, this is summarizing, reporting.

Beauty and the beast

Both querying and aggregating are crucial, but one is cat'ly better behaved.

- Querying is part of a larger story called *data migration*.
- Given two categories (DB schemas) $\mathscr{C}, \mathscr{D}$, a data migration functor...
- ... $\mathscr{C} \triangleleft\!\!\longrightarrow\!\!\triangleleft \mathscr{D}$ is a parametric right adjoint $\mathscr{D}$-**Set** $\to \mathscr{C}$-**Set**.
- These have nice characterizations, some of which we'll discuss, and...
- ... have been implemented in open-source categorically-minded code.

Beauty and the beast

Both querying and aggregating are crucial, but one is cat'ly better behaved.

- Querying is part of a larger story called *data migration*.
- Given two categories (DB schemas) $\mathcal{C}, \mathcal{D}$, a data migration functor...
- ... $\mathcal{C} \mathrel{\triangleleft\!\!\longleftarrow\!\!\triangleleft} \mathcal{D}$ is a parametric right adjoint $\mathcal{D}$-**Set** $\to \mathcal{C}$-**Set**.
- These have nice characterizations, some of which we'll discuss, and...
- ... have been implemented in open-source categorically-minded code.

In contrast, aggregation has seemingly not received a categ'al formulation.

- It's not even clear what sort of properties are desirable.
- For example, aggregation is not natural wrt. copresheaf morphisms.
- Consider: what is preserved by a commutative square $f' \to f$?

$$
\begin{tikzcd}
P' \arrow[r] \arrow[d, "f'"'] & P \arrow[r, "s"] \arrow[d, "f"'] & \mathbb{R} \\
B' \arrow[r] & B \arrow[ur, dashed, "(\mathtt{sum}\ s)_f"'] &
\end{tikzcd}
$$

Beauty and the beast

Both querying and aggregating are crucial, but one is cat'ly better behaved.

- Querying is part of a larger story called *data migration*.
- Given two categories (DB schemas) $\mathcal{C}, \mathcal{D}$, a data migration functor...
- ... $\mathcal{C} \triangleleft\!\!\longrightarrow\!\!\triangleleft\; \mathcal{D}$ is a parametric right adjoint $\mathcal{D}$-**Set** $\to$ $\mathcal{C}$-**Set**.
- These have nice characterizations, some of which we'll discuss, and...
- ... have been implemented in open-source categorically-minded code.

In contrast, aggregation has seemingly not received a categ'al formulation.

- It's not even clear what sort of properties are desirable.
- For example, aggregation is not natural wrt. copresheaf morphisms.
- Consider: what is preserved by a commutative square $f' \to f$?

$$\begin{tikzcd}
P' \arrow[r] \arrow[d, "f'"'] & P \arrow[r, "s"] \arrow[d, "f"'] & \mathbb{R} \\
B' \arrow[r] & B \arrow[ur, dashed, "(\mathrm{sum}\ s)_f"'] &
\end{tikzcd}$$

Can you make a match between beauty and the beast? "Be my guest!"

Poly-amory: could one category be enough?

I've lately been totally enamored with **Poly** and its comonoids $\mathbb{C}\mathbf{at}^{\sharp}$.

- First, it is an excellent setting for thinking about things I care about.
  - It is a natural setting for interacting dynamical systems.
  - And thanks to the results of Ahman-Uustalu and Garner...
  - ...it is a natural setting for databases and data migration.
  - Both of these seem relevant when accounting for sense-making.

Poly-amory: could one category be enough?

I've lately been totally enamored with **Poly** and its comonoids $\mathbb{C}\mathbf{at}^\sharp$.

- First, it is an excellent setting for thinking about things I care about.
  - It is a natural setting for interacting dynamical systems.
  - And thanks to the results of Ahman-Uustalu and Garner...
  - ...it is a natural setting for databases and data migration.
  - Both of these seem relevant when accounting for sense-making.
- Second, **Poly** has loads and loads of structure.
  - Coproducts and products that agree with usual polynomial arithmetic;
  - All limits and colimits;
  - At least three orthogonal factorization systems;
  - A symmetric monoidal structure $\otimes$ distributing over $+$;
  - A cartesian closure q^p and monoidal closure $[p, q]$ for $\otimes$;
  - Another nonsymmetric monoidal structure $\triangleleft$ that's duoidal with $\otimes$;
  - A left $\triangleleft$-coclosure $\left[\begin{smallmatrix}-\\-\end{smallmatrix}\right]$, meaning $\mathbf{Poly}(p, q \triangleleft r) \cong \mathbf{Poly}(\left[\begin{smallmatrix}r\\p\end{smallmatrix}\right], q)$;
  - An indexed right $\triangleleft$-coclosure (Myers?), i.e. $\mathbf{Poly}(p, q \triangleleft r) \cong \sum_{f\colon\ p(1)\to q(1)} \mathbf{Poly}(p \overset{f}{\frown} q, r)$;
  - An indexed right $\otimes$-coclosure (Niu?), i.e. $\mathbf{Poly}(p, q \otimes r) \cong \sum_{f\colon\ p(1)\to q(1)} \mathbf{Poly}(p \overset{f}{\nearrow} q, r)$;
  - At least eight monoidal structures in total;
  - $\triangleleft$-monoids generalize plain operads;
  - $\triangleleft$-comonoids are exactly categories; bicomodules are data migrations. This is $\mathbb{C}\mathbf{at}^\sharp$.
  - For the above and more, see "A reference for categorical structures on **Poly**", arXiv: 2202.00534

It was love at first sight. I'm committed to solving problems as a team.

Aggregation poses a "purity of methods" problem

"Solving aggregation" is not well-defined.

- Given a map $E \to B$ and a map $E \to \mathbb{R}$, you "just integrate".
- It's hard to know what problem needs solving.

Aggregation poses a "purity of methods" problem

"Solving aggregation" is not well-defined.

- Given a map $E \to B$ and a map $E \to \mathbb{R}$, you "just integrate".
- It's hard to know what problem needs solving.

According to Detlefsen & Arana, "purity of methods" has a long tradition.

- Aristotle, Newton, Lagrange, Gauss, Bolzano, Frege,... all sought it.
- Erdös wanted a non-$\mathbb{C}$ proof of Hadamard's prime number thm ($\frac{n}{\log n}$).
- Bolzano phrased it as searching for "a thorough way of thinking."

Aggregation poses a "purity of methods" problem

"Solving aggregation" is not well-defined.

- Given a map $E \to B$ and a map $E \to \mathbb{R}$, you "just integrate".
- It's hard to know what problem needs solving.

According to Detlefsen & Arana, "purity of methods" has a long tradition.

- Aristotle, Newton, Lagrange, Gauss, Bolzano, Frege,... all sought it.
- Erdös wanted a non-$\mathbb{C}$ proof of Hadamard's prime number thm ($\frac{n}{\log n}$).
- Bolzano phrased it as searching for "a thorough way of thinking."

So this suggests a ways forward.

- Since **Poly** is great for thinking about data migration (as I'll discuss)...
- ...it is a "purity of methods" issue to get aggregation into **Poly** as well.
- So the goal is to give an account of aggregation using...
- ...only monoidal/universal structures available in the **Poly** ecosystem.
- Pursuing it led to several new structures, which I'll tell you about.

Plan of the talk

Now that I've introduced the topic, here's the plan for my remaining time.

- Give background on **Poly**, its monoidal closure and mon'l coclosure.
- Discuss $\mathbb{C}\mathbf{at}^\sharp = \mathbb{C}\mathbf{omon}(\mathbf{Poly})$, natural home of data migration.
- Show how aggregation-useful structures on **Poly** generalize to $\mathbb{C}\mathbf{at}^\sharp$.
- Explain aggregation with the structures we've explored.
- Conclude with a summary.

Outline

Poly: coproducts of representables Set $\to$ Set

A polynomial functor is a coproduct of representables **Set** $\to$ **Set**:

- For any set E, denote the functor it represents by $y^E \coloneqq \mathbf{Set}(E, -)$.
- E.g. $y = y^1$ is identity, $y^0 = 1$ is constant, and $y^E(1) \cong 1$ for any E.

Poly: coproducts of representables Set → Set

A polynomial functor is a coproduct of representables **Set** → **Set**:

- For any set E, denote the functor it represents by $\mathcal{y}^E \coloneqq \mathbf{Set}(E, -)$.
- E.g. $\mathcal{y} = \mathcal{y}^1$ is identity, $\mathcal{y}^0 = 1$ is constant, and $\mathcal{y}^E(1) \cong 1$ for any E.
- A *polynomial* is a disjoint union of representables $p \cong \sum_{b \in B} \mathcal{y}^{E_b}$.
- Note that $p(1) \cong B$ so, we can denote polynomials as follows:

$$p \coloneqq \sum_{I \in p(1)} \mathcal{y}^{p[I]}$$

Poly: coproducts of representables Set → Set

A polynomial functor is a coproduct of representables **Set** → **Set**:

- For any set E, denote the functor it represents by $y^E \coloneqq \mathbf{Set}(E, -)$.
- E.g. $y = y^1$ is identity, $y^0 = 1$ is constant, and $y^E(1) \cong 1$ for any E.
- A *polynomial* is a disjoint union of representables $p \cong \sum_{b \in B} y^{E_b}$.
- Note that $p(1) \cong B$ so, we can denote polynomials as follows:

$$p \coloneqq \sum_{I \in p(1)} y^{p[I]}$$

Morphisms $p \xrightarrow{\varphi} q$ are just natural transformations $\begin{tikzcd} \mathbf{Set} \arrow[r, bend left, "p"] \arrow[r, bend right, "q"'] \arrow[r, phantom, "\Downarrow\varphi"] & \mathbf{Set} \end{tikzcd}$

- Combinatorially, a map $\varphi \colon p \to q$ can be given in two parts:
- A function $\varphi_1 \colon p(1) \to q(1)$ "forward on positions" and...
- ...for each $I \in p(1)$, a function $q[\varphi_1 I] \to p[I]$ "backward on directions"

Poly: coproducts of representables $\mathbf{Set} \to \mathbf{Set}$

A polynomial functor is a coproduct of representables $\mathbf{Set} \to \mathbf{Set}$:

- For any set E, denote the functor it represents by $y^E \coloneqq \mathbf{Set}(E, -)$.
- E.g. $y = y^1$ is identity, $y^0 = 1$ is constant, and $y^E(1) \cong 1$ for any E.
- A *polynomial* is a disjoint union of representables $p \cong \sum_{b \in B} y^{E_b}$.
- Note that $p(1) \cong B$ so, we can denote polynomials as follows:

$$p \coloneqq \sum_{I \in p(1)} y^{p[I]}$$

Morphisms $p \xrightarrow{\varphi} q$ are just natural transformations $\begin{tikzcd} \mathbf{Set} \arrow[r, bend left, "p"] \arrow[r, bend right, "q"'] \arrow[r, phantom, "\Downarrow\varphi"] & \mathbf{Set} \end{tikzcd}$

- Combinatorially, a map $\varphi \colon p \to q$ can be given in two parts:
- A function $\varphi_1 \colon p(1) \to q(1)$ "forward on positions" and...
- ...for each $I \in p(1)$, a function $q[\varphi_1 I] \to p[I]$ "backward on directions"

A polynomial can be viewed as a functor or just as a combinatorial object.

- Polynomials can be viewed as functors; this is like "querying".
- The functor p "migrates data", sending $X \in \mathbf{Set}$ to $p(X) \in \mathbf{Set}$.
- But it's often helpful just to think of p as a data structure.

Dirichlet product $\otimes$ and its closure

Poly admits many monoidal structures, e.g. coproduct and product $(+, \times)$.

- Among the most useful is Dirichlet product $\otimes$; its unit is y.
- If you think of a polynomial p as a bundle, $\left(\sum_{I \in p(1)} p[I]\right) \to p(1)$...
- ...then $p \otimes q$ is just product of base and total spaces for $p, q \in$ **Poly**.

$$p \otimes q \cong \sum_{(I,J) \in p(1) \times q(1)} y^{p[I] \times q[J]}$$

Dirichlet product $\otimes$ and its closure

Poly admits many monoidal structures, e.g. coproduct and product $(+, \times)$.

- Among the most useful is Dirichlet product $\otimes$; its unit is y.
- If you think of a polynomial p as a bundle, $\left(\sum_{I \in p(1)} p[I]\right) \to p(1)$...
- ...then $p \otimes q$ is just product of base and total spaces for $p, q \in \textbf{Poly}$.

$$p \otimes q \cong \sum_{(I,J) \in p(1) \times q(1)} y^{p[I] \times q[J]}$$

The $\otimes$-structure has a closure: $\textbf{Poly}(p \otimes q, r) \cong \textbf{Poly}(p, [q, r])$:

$$[q, r] \cong \sum_{\varphi \in \textbf{Poly}(q,r)} y^{\sum\limits_{J \in q(1)} r[\varphi_1 J]}$$

Dirichlet product $\otimes$ and its closure

Poly admits many monoidal structures, e.g. coproduct and product $(+, \times)$.

- Among the most useful is Dirichlet product $\otimes$; its unit is y.
- If you think of a polynomial p as a bundle, $\left(\sum_{I \in p(1)} p[I]\right) \to p(1)$...
- ...then $p \otimes q$ is just product of base and total spaces for $p, q \in \textbf{Poly}$.

$$p \otimes q \cong \sum_{(I,J) \in p(1) \times q(1)} y^{p[I] \times q[J]}$$

The $\otimes$-structure has a closure: $\textbf{Poly}(p \otimes q, r) \cong \textbf{Poly}(p, [q, r])$:

$$[q, r] \cong \sum_{\varphi \in \textbf{Poly}(q,r)} y^{\sum_{J \in q(1)} r[\varphi_1 J]}$$

Like so much in **Poly**, both $\otimes$ and $[-, -]$ generalize to $\mathbb{C}\mathbf{at}^\sharp$.

- The polynomial y is a *dualizing object*: for any $A \in \textbf{Set}$...
- ...we have isomorphisms $[Ay, y] \cong y^A$ and $[y^A, y] \cong Ay$.
- We write $\overline{Ay} \cong y^A$. This generalizes to an important part of our story.

Substitution product $\triangleleft$ and its coclosure

The other important monoidal product is called *substitution* or *composition*.

- The composite $p \triangleleft q := p \circ q$ of polynomial functors is a polynomial.
- E.g. if $p = y^2$ and $q = y + 1$, then $p \triangleleft q \cong y^2 + 2y + 1$.
- (There are many reasons for $\triangleleft$ instead of $\circ$. One is that we want to reserve $\circ$ for morphisms; $\triangleleft$ is "composing" objects!)
- This monoidal product $\triangleleft$ preserves equalizers in both variables.
- It and $\otimes$ are duoidal: $(p_1 \triangleleft p_2) \otimes (q_1 \triangleleft q_2) \to (p_1 \otimes q_1) \triangleleft (p_2 \otimes q_2)$.

Substitution product $\triangleleft$ and its coclosure

The other important monoidal product is called *substitution* or *composition*.

- The composite $p \triangleleft q := p \circ q$ of polynomial functors is a polynomial.
- E.g. if $p = y^2$ and $q = y + 1$, then $p \triangleleft q \cong y^2 + 2y + 1$.
- (There are many reasons for $\triangleleft$ instead of $\circ$. One is that we want to reserve $\circ$ for morphisms; $\triangleleft$ is "composing" objects!)
- This monoidal product $\triangleleft$ preserves equalizers in both variables.
- It and $\otimes$ are duoidal: $(p_1 \triangleleft p_2) \otimes (q_1 \triangleleft q_2) \to (p_1 \otimes q_1) \triangleleft (p_2 \otimes q_2)$.

In fact, $\triangleleft$ has a right coclosure (Myers?) and an indexed left coclosure.

- We'll only need the former, $\mathbf{Poly}(p, q \triangleleft p') \cong \mathbf{Poly}\left(\begin{bmatrix} p' \\ p \end{bmatrix}, q\right)$.
- For any $p \in \mathbf{Poly}$ the polynomial $\begin{bmatrix} p \\ p \end{bmatrix}$ is a $\triangleleft$-comonoid (Meyers).
- We'll see that $\triangleleft$-comonoids are categories; which one is this?
- It's the *full internal subcat'y of* $\mathbf{Set}^{\mathrm{op}}$ (Jacobs) spanned by p-fibers.

Substitution product $\triangleleft$ and its coclosure

The other important monoidal product is called *substitution* or *composition*.

- The composite $p \triangleleft q := p \circ q$ of polynomial functors is a polynomial.
- E.g. if $p = y^2$ and $q = y + 1$, then $p \triangleleft q \cong y^2 + 2y + 1$.
- (There are many reasons for $\triangleleft$ instead of $\circ$. One is that we want to reserve $\circ$ for morphisms; $\triangleleft$ is "composing" objects!)
- This monoidal product $\triangleleft$ preserves equalizers in both variables.
- It and $\otimes$ are duoidal: $(p_1 \triangleleft p_2) \otimes (q_1 \triangleleft q_2) \to (p_1 \otimes q_1) \triangleleft (p_2 \otimes q_2)$.

In fact, $\triangleleft$ has a right coclosure (Myers?) and an indexed left coclosure.

- We'll only need the former, $\mathbf{Poly}(p, q \triangleleft p') \cong \mathbf{Poly}\left(\begin{bmatrix} p' \\ p \end{bmatrix}, q\right)$.
- For any $p \in \mathbf{Poly}$ the polynomial $\begin{bmatrix} p \\ p \end{bmatrix}$ is a $\triangleleft$-comonoid (Meyers).
- We'll see that $\triangleleft$-comonoids are categories; which one is this?
- It's the *full internal subcat'y of* $\mathbf{Set}^{\mathrm{op}}$ (Jacobs) spanned by p-fibers.

We'll rely heavily on this in a special case: $u = \mathrm{List} = \sum_{N \in \mathbb{N}} y^N$.

- Then $\begin{bmatrix} u \\ u \end{bmatrix}$ is a skeleton of $\mathbf{Fin}^{\mathrm{op}}$.
- Later we'll get its opposite as the dual, $\overline{\begin{bmatrix} u \\ u \end{bmatrix}} \simeq \mathbf{Fin}$.

Outline

$\mathbb{C}\mathbf{at}^\sharp := \mathbb{C}\mathbf{omon}(\mathbf{Poly})$

Shulman: if equipment $\mathbb{P}$ has good equalizers, $\mathbb{C}\mathbf{omon}(\mathbb{P})$ is an equipment.

- An *equipment* is a kind of double cat'y, where 2-cells can be "cartesian."
- Any bicat'y—and hence any monoidal category—is one, called *globular*.
- So $(\mathbf{Poly}, y, \triangleleft)$ is a equipment! It happens to be vertically-trivial.
- We said **Poly** has "good equalizers": i.e. they are preserved by $\triangleleft$.
- Shulman tells us that $\mathbb{C}\mathbf{at}^\sharp := \mathbb{C}\mathbf{omon}(\mathbf{Poly})$ is also an equipment.

$\mathbb{C}\mathbf{at}^\sharp := \mathbb{C}\mathbf{omon}(\mathbf{Poly})$

Shulman: if equipment $\mathbb{P}$ has good equalizers, $\mathbb{C}\mathbf{omon}(\mathbb{P})$ is an equipment.

- An *equipment* is a kind of double cat'y, where 2-cells can be "cartesian."
- Any bicat'y—and hence any monoidal category—is one, called *globular*.
- So $(\mathbf{Poly}, y, \triangleleft)$ is a equipment! It happens to be vertically-trivial.
- We said **Poly** has "good equalizers": i.e. they are preserved by $\triangleleft$.
- Shulman tells us that $\mathbb{C}\mathbf{at}^\sharp := \mathbb{C}\mathbf{omon}(\mathbf{Poly})$ is also an equipment.

Ahman-Uustalu: the objects of $\mathbb{C}\mathbf{omon}(\mathbf{Poly})$ are categories.

- A comonoid in $(\mathbf{Poly}, y, \triangleleft)$ consists of (c, ϵ, δ) where $c \in \mathbf{Poly}$ and...
- ...$\epsilon\colon c \to y$ and $\delta\colon c \to c \triangleleft c$ are (counital & coassoc'tive) maps.
- This turns out to force $c \cong \sum_{a \in \mathrm{Ob}(\mathcal{C})} y^{\mathcal{C}[a]}$ for some category $\mathcal{C}$,...
- ...where $\mathcal{C}[a] := \sum_{a' \in \mathrm{Ob}(\mathcal{C})} \mathrm{Hom}_{\mathcal{C}}(a, a')$. Say c is $\mathcal{C}$'s *outfacing poly'l*.
- Then ϵ gives identities and δ gives codomains and composition.

$\mathbb{C}\mathbf{at}^\sharp := \mathbb{C}\mathbf{omon}(\mathbf{Poly})$

Shulman: if equipment $\mathbb{P}$ has good equalizers, $\mathbb{C}\mathbf{omon}(\mathbb{P})$ is an equipment.

- An *equipment* is a kind of double cat'y, where 2-cells can be "cartesian."
- Any bicat'y—and hence any monoidal category—is one, called *globular*.
- So $(\mathbf{Poly}, y, \triangleleft)$ is a equipment! It happens to be vertically-trivial.
- We said **Poly** has "good equalizers": i.e. they are preserved by $\triangleleft$.
- Shulman tells us that $\mathbb{C}\mathbf{at}^\sharp := \mathbb{C}\mathbf{omon}(\mathbf{Poly})$ is also an equipment.

Ahman-Uustalu: the objects of $\mathbb{C}\mathbf{omon}(\mathbf{Poly})$ are categories.

- A comonoid in $(\mathbf{Poly}, y, \triangleleft)$ consists of (c, ϵ, δ) where $c \in \mathbf{Poly}$ and...
- ...$\epsilon \colon c \to y$ and $\delta \colon c \to c \triangleleft c$ are (counital & coassoc'tive) maps.
- This turns out to force $c \cong \sum_{a \in \mathrm{Ob}(\mathcal{C})} y^{\mathcal{C}[a]}$ for some category $\mathcal{C}$,...
- ...where $\mathcal{C}[a] := \sum_{a' \in \mathrm{Ob}(\mathcal{C})} \mathrm{Hom}_{\mathcal{C}}(a, a')$. Say c is $\mathcal{C}$'s *outfacing poly'l*.
- Then ϵ gives identities and δ gives codomains and composition.

Garner: horizontal morphisms of $\mathbb{C}\mathbf{omon}(\mathbf{Poly})$ are data migrations.

- He didn't say it that way. He said that a bicomodule $\mathcal{C} \triangleleft\!\xrightarrow{p}\!\triangleleft\ \mathcal{D}$...
- ...is a parametric right adjoint $\mathbf{Set}^{\mathcal{D}} \to \mathbf{Set}^{\mathcal{C}}$. I'll explain soon.
- I denote the cat'y of (c, d)-bicomodules by c-$\mathbf{Set}[d]$.

$\mathbb{P}$olyFun and $\mathbb{S}$pan as subequipments of $\mathbb{C}\mathbf{at}^\sharp$

To understand $\mathbb{C}\mathbf{at}^\sharp := \mathbb{C}\mathbf{omon}(\mathbf{Poly})$, let's start with something familiar.

- Gambino-Kock showed that sets, functions, and multi-variate poly's...
- ...form an equipment, called $\mathbb{P}\mathbf{olyFun}$. (And similarly for arbitrary LCC cat'y in place of **Set**).

$\mathbb{P}$olyFun and $\mathbb{S}$pan as subequipments of $\mathbb{C}\mathbf{at}^\sharp$

To understand $\mathbb{C}\mathbf{at}^\sharp := \mathbb{C}\mathbf{omon}(\mathbf{Poly})$, let's start with something familiar.

- Gambino-Kock showed that sets, functions, and multi-variate poly's...
- ...form an equipment, called $\mathbb{P}\mathbf{olyFun}$. (And similarly for arbitrary LCC cat'y in place of **Set**).
- It sits inside $\mathbb{C}\mathbf{at}^\sharp$ as the full subequipment spanned by discrete caty's.
- Discrete caty's are those whose outfacing poly'l is linear, $I\mathcal{y}$ for $I \in \mathbf{Set}$.

$$
\begin{tikzcd}
I \arrow[dd] & E \arrow[l] \arrow[r] & B \arrow[r] \arrow[d, equal] & J \arrow[dd] \\
 & E' \times_{B'} B \arrow[u, "\varphi^\sharp"] \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner"] & B \arrow[d, "\varphi_1"] & \\
I' & E' \arrow[l] \arrow[r] & B' \arrow[r] & J'
\end{tikzcd}
\quad \leftrightsquigarrow \quad
\begin{tikzcd}
I\mathcal{y} \arrow[d] & J\mathcal{y} \arrow[l, "p"'] \arrow[d] \arrow[dl, phantom, "\Downarrow \varphi"] \\
I'\mathcal{y} & J'\mathcal{y} \arrow[l, "p'"]
\end{tikzcd}
$$

All the "activity" is subsumed under the def'n of comonoid, comodule.

$\mathbb{P}$olyFun and $\mathbb{S}$pan as subequipments of $\mathbb{C}\mathbf{at}^\sharp$

To understand $\mathbb{C}\mathbf{at}^\sharp := \mathbb{C}\mathbf{omon}(\mathbf{Poly})$, let's start with something familiar.

- Gambino-Kock showed that sets, functions, and multi-variate poly's...
- ...form an equipment, called $\mathbb{P}\mathbf{olyFun}$. (And similarly for arbitrary LCC cat'y in place of **Set**).
- It sits inside $\mathbb{C}\mathbf{at}^\sharp$ as the full subequipment spanned by discrete caty's.
- Discrete caty's are those whose outfacing poly'l is linear, Iy for $I\in\mathbf{Set}$.

$$\begin{tikzcd}
I \arrow[ddd] & E \arrow[l] \arrow[rr] & & B \arrow[r] \arrow[d, equal] & J \arrow[ddd] \\
& E' \times_{B'} B \arrow[u, "\varphi^\sharp"] \arrow[rr] \arrow[dd] \arrow[ddrr, phantom, "\lrcorner", very near start] & & B \arrow[dd, "\varphi_1"] & \\
& & & & \\
I' & E' \arrow[l] \arrow[rr] & & B' \arrow[r] & J'
\end{tikzcd}
\quad \leftrightsquigarrow \quad
\begin{tikzcd}
Iy \arrow[d] \arrow[dr, phantom, "\Downarrow \varphi"] & Jy \arrow[l, "p"'] \arrow[d] \\
I'y & J'y \arrow[l, "p'"]
\end{tikzcd}$$

All the "activity" is subsumed under the def'n of comonoid, comodule.

The usual double cat'y of spans sits inside: $\mathbb{S}\mathbf{pan} \subseteq \mathbb{P}\mathbf{olyFun} \subseteq \mathbb{C}\mathbf{at}^\sharp$.

- $\mathbb{S}\mathbf{pan} \subseteq \mathbb{C}\mathbf{at}^\sharp$ is the subequipment where every poly is linear!
- Discrete carrier makes: a cat'y be discrete, a bicomodule be a span.

Database schemas and Duc-queries

We can get more intuition for $\mathbb{C}\mathbf{at}^\sharp$ by thinking about databases.

- The indexing cat'y for graphs is $\mathcal{G} := \boxed{\bullet \rightrightarrows \bullet}$, carried by

Database schemas and Duc-queries

We can get more intuition for $\mathbb{C}\mathbf{at}^\sharp$ by thinking about databases.

- The indexing cat'y for graphs is $\mathcal{G} := \boxed{\bullet \rightrightarrows \bullet}$, carried by $g := y^3 + y$.
- Bicomodules $c \triangleleft\!\xleftarrow{X}\!\triangleleft\ 0$ can be ident'd with copresheaves $c \xrightarrow{X} \mathbf{Set}$.
- So a graph is just a bicomodule $g \triangleleft\!\xleftarrow{X}\!\triangleleft\ 0$. Call X a g-set or $\mathcal{G}$-set.
- Think of $\mathcal{G}$ as a database *schema*, an arrangement for sets, and...
- ...think of $X : \mathcal{G} \to \mathbf{Set}$ as a $\mathcal{G}$-*instance*, some sets so-arranged.

Database schemas and Duc-queries

We can get more intuition for $\mathbb{C}\mathbf{at}^\sharp$ by thinking about databases.

- The indexing cat'y for graphs is $\mathcal{G} := \boxed{\bullet \rightrightarrows \bullet}$, carried by $g := y^3 + y$.
- Bicomodules $c \triangleleft\!\xleftarrow{X}\!\triangleleft\ 0$ can be ident'd with copresheaves $c \xrightarrow{X} \mathbf{Set}$.
- So a graph is just a bicomodule $g \triangleleft\!\xleftarrow{X}\!\triangleleft\ 0$. Call X a g-set or $\mathcal{G}$-set.
- Think of $\mathcal{G}$ as a database *schema*, an arrangement for sets, and...
- ...think of $X\colon \mathcal{G} \to \mathbf{Set}$ as a $\mathcal{G}$-*instance*, some sets so-arranged.

We can move data between database schemas using duc-queries.

- The cat'y of *conjunctive queries on* $\mathcal{C}$ is $(\mathcal{C}\text{-}\mathbf{Set})^{\text{op}}$. Idea:
- For $Q \in \mathcal{C}\text{-}\mathbf{Set}$ we have a functor $\mathcal{C}\text{-}\mathbf{Set}(Q, -)\colon \mathcal{C}\text{-}\mathbf{Set} \to \mathbf{Set}$.
- We think of Q as a query: find me all Q-shapes in $-$. Contravariant in Q.
- E.g. there's a graph Q_n for which $\mathcal{G}\text{-}\mathbf{Set}(Q_n, -)$ returns length-n paths.
- If we want all paths, we need a disjoint union of conjunctive query's.

Database schemas and Duc-queries

We can get more intuition for $\mathbb{C}\mathbf{at}^\sharp$ by thinking about databases.

- The indexing cat'y for graphs is $\mathcal{G} := \boxed{\bullet \rightrightarrows \bullet}$, carried by $g := y^3 + y$.
- Bicomodules $c \triangleleft\!\xrightarrow{X}\!\triangleleft\ 0$ can be ident'd with copresheaves $c \xrightarrow{X} \mathbf{Set}$.
- So a graph is just a bicomodule $g \triangleleft\!\xrightarrow{X}\!\triangleleft\ 0$. Call X a g-set or $\mathcal{G}$-set.
- Think of $\mathcal{G}$ as a database *schema*, an arrangement for sets, and...
- ...think of $X\colon \mathcal{G} \to \mathbf{Set}$ as a $\mathcal{G}$-*instance*, some sets so-arranged.

We can move data between database schemas using duc-queries.

- The cat'y of *conjunctive queries on* $\mathcal{C}$ is $(\mathcal{C}\text{-}\mathbf{Set})^{\mathrm{op}}$. Idea:
- For $Q \in \mathcal{C}\text{-}\mathbf{Set}$ we have a functor $\mathcal{C}\text{-}\mathbf{Set}(Q,-)\colon \mathcal{C}\text{-}\mathbf{Set} \to \mathbf{Set}$.
- We think of Q as a query: find me all Q-shapes in $-$. Contravariant in Q.
- E.g. there's a graph Q_n for which $\mathcal{G}\text{-}\mathbf{Set}(Q_n,-)$ returns length-n paths.
- If we want all paths, we need a disjoint union of conjunctive query's.

A data migration (bicomod) $\mathcal{C} \triangleleft\!\xrightarrow{p}\!\triangleleft\ \mathcal{D}$ is a $\mathcal{C}$-indexed duc-query on $\mathcal{D}$.

- It functorially assigns a duc-query on $\mathcal{D}$ to each $c \in \mathcal{C}$.
- Given a $\mathcal{D}$-instance $\mathcal{D} \triangleleft\!\longrightarrow\!\triangleleft\ 0$, composition returns a $\mathcal{C}$-instance.

Adjoint prafunctors

Data migrations = Bicomodules = *parametric right adjoint* functors.

- A bicomodule $c \overset{p}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} d$ is a prafunctor d-**Set** $\to$ c-**Set**.
- It is also a c-indexed duc-query on d. This can be helpful.
- E.g., a profunctor $c^{\mathrm{op}} \times d \to \mathbf{Set}$ is a special case of prafunctor.
- It can be identified with a c-indexed *conjunctive* query on d, no sums.

Adjoint prafunctors

Data migrations = Bicomodules = *parametric right adjoint* functors.

- A bicomodule $c \overset{p}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} d$ is a prafunctor d-**Set** $\to$ c-**Set**.
- It is also a c-indexed duc-query on d. This can be helpful.
- E.g., a profunctor $c^{\mathsf{op}} \times d \to \mathbf{Set}$ is a special case of prafunctor.
- It can be identified with a c-indexed *conjunctive* query on d, no sums.

When is a prafunctor $c \overset{p}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} d$ a right adjoint in $\mathbb{C}\mathbf{at}^\sharp$?

- Two conditions: it is a right adjoint functor d-**Set** $\to$ c-**Set** and...
- ...the associated left adjoint c-**Set** $\to$ d-**Set** is also a prafunctor.
- For example for any functor $F\colon c \to d$, we have $\Delta_F \dashv \Pi_F$.
- For $c \overset{p}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} d$, I denote its left adjoint by $d \overset{p^\dagger}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} c$.

Adjoint prafunctors

Data migrations = Bicomodules = *parametric right adjoint* functors.

- A bicomodule $c \xleftarrow{p} d$ is a prafunctor d-**Set** $\to$ c-**Set**.
- It is also a c-indexed duc-query on d. This can be helpful.
- E.g., a profunctor $c^{\mathrm{op}} \times d \to \mathbf{Set}$ is a special case of prafunctor.
- It can be identified with a c-indexed *conjunctive* query on d, no sums.

When is a prafunctor $c \xleftarrow{p} d$ a right adjoint in $\mathbb{C}\mathbf{at}^\sharp$?

- Two conditions: it is a right adjoint functor d-**Set** $\to$ c-**Set** and...
- ...the associated left adjoint c-**Set** $\to$ d-**Set** is also a prafunctor.
- For example for any functor $F\colon c \to d$, we have $\Delta_F \dashv \Pi_F$.
- For $c \xleftarrow{p} d$, I denote its left adjoint by $d \xleftarrow{p^\dagger} c$.

In the subcategory $\mathbb{P}\mathbf{olyFun} = \mathbb{C}\mathbf{at}^\sharp_{\mathrm{disc}}$, adjoints are easy to characterize:

- Left adjoints are those p with linear carrier (spans), and...
- ...right adjoints are the profunctors, i.e. c-indexed conjunctive queries.

External and internal $\otimes$

The equipment $\mathbb{C}\mathbf{at}^\sharp$ has lots of structure, e.g. it is monoidal.

- There is a double functor $\otimes \colon \mathbb{C}\mathbf{at}^\sharp \times \mathbb{C}\mathbf{at}^\sharp \to \mathbb{C}\mathbf{at}^\sharp$.
- It is an (external) symmetric monoidal structure on $\mathbb{C}\mathbf{at}^\sharp$.
- On objects, $c \otimes d$ is the usual product of categories.

External and internal $\otimes$

The equipment $\mathbb{C}\mathbf{at}^\sharp$ has lots of structure, e.g. it is monoidal.

- There is a double functor $\otimes\colon \mathbb{C}\mathbf{at}^\sharp \times \mathbb{C}\mathbf{at}^\sharp \to \mathbb{C}\mathbf{at}^\sharp$.
- It is an (external) symmetric monoidal structure on $\mathbb{C}\mathbf{at}^\sharp$.
- On objects, $c \otimes d$ is the usual product of categories.

It also has local $\otimes$-monoidal structures.

- For any $c, d \in \mathbb{C}\mathbf{at}^\sharp$ the cat'y c-$\mathbf{Set}[d]$ has an induced $\otimes$-structure.
- That is, for any two bicomodules $c \overset{p,q}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} d$, there is...
- ...a bicomodule $c \overset{p\ {}_c\!\otimes_d\ q}{\triangleleft\!\!\longrightarrow\!\!\triangleleft} d$. The local unit is $c(1)y^{d(1)}$.
- These fit together "duoidally":

$$\begin{tikzcd} c_0 \arrow[r, shift left, "p_1"] \arrow[r, shift right, "q_1"'] & c_1 \arrow[r, shift left, "p_2"] \arrow[r, shift right, "q_2"'] & c_2 \end{tikzcd} \quad \rightsquigarrow \quad \begin{tikzcd} c_0 \arrow[rr, bend left, "(p_1 \triangleleft_{c_1} p_2)\ {}_{c_0}\!\otimes_{c_2} (q_1 \triangleleft_{c_1} q_2)"] \arrow[rr, bend right, "(p_1\ {}_{c_0}\!\otimes_{c_1} q_1) \triangleleft_{c_1} (p_2\ {}_{c_1}\!\otimes_{c_2} q_2)"'] \arrow[rr, phantom, "\Downarrow \text{duoid}"] && c_2 \end{tikzcd}$$

Local closures and dualizing object

The local $\otimes$-structures have closures.

- That is, for $p, q, r \in c$-$\mathbf{Set}[d]$, there is a natural isomorphism

$$c\text{-}\mathbf{Set}[d](p \, {}_c\!\otimes_d \, q \, , \, r) \cong c\text{-}\mathbf{Set}[d](p \, , \, {}_c[q, r]_d)$$

- Wanted: other equip's with local (duoidal) monoidal-closed structure.

Local closures and dualizing object

The local $\otimes$-structures have closures.

- That is, for $p, q, r \in c\text{-}\mathbf{Set}[d]$, there is a natural isomorphism

$$c\text{-}\mathbf{Set}[d](p\,{}_c\!\otimes_d\, q\,,\ r) \cong c\text{-}\mathbf{Set}[d](p\,,\ {}_c[q, r]_d)$$

- Wanted: other equip's with local (duoidal) monoidal-closed structure.

For any sets C, D, there's a dualizing object in $C\text{-}\mathbf{Set}[D]{=}\mathbf{Poly}_{GK}(C, D)$.

- It's the terminal span, $C \leftarrow (C \times D) \rightarrow D$, i.e. $C\mathcal{y} \overset{CD\mathcal{y}}{\triangleleft\!\!-\!\!\!-\!\!\!-\!\!\triangleleft} D\mathcal{y}$.
- Calling it $\perp := CD\mathcal{y}$, the functor $\overline{\,\cdot\,} := {}_C[-, \perp]_D$ provides a duality...
- If $p \in C\text{-}\mathbf{Set}[D]$ is linear then $[p, \perp]$ is conjunctive, and vice versa.
- In particular $\overline{\overline{p}} \cong p$ for any linear or conjunctive p.
- It generalizes $[A\mathcal{y}, \mathcal{y}] \cong \mathcal{y}^A$ from earlier.

Transposing a span, "oppositing" a category

The idea of $\overline{p}$ is that it transforms $\Sigma_F \circ \Delta_G$ into $\Pi_F \circ \Delta_G$.

- That's not its adjoint!
- The adjoint of $\Sigma_F \circ \Delta_G$ is $\Pi_G \circ \Delta_F$.
- So what is the adjoint of the dual or the dual of the adjoint?
- Answer: $\Sigma_F \circ \Delta_G \;\mapsto\; \Pi_F \circ \Delta_G \;\mapsto\; \Sigma_G \circ \Delta_F$. (The other works too.)

On the level of spans, this is the transpose!

- The transpose operation is a composite of two more primitive ones.
- This doesn't happen within $\mathbb{S}$**pan**; kind of like contour integrals.

Transposing a span, "oppositing" a category

The idea of $\overline{p}$ is that it transforms $\Sigma_F \circ \Delta_G$ into $\Pi_F \circ \Delta_G$.

- That's not its adjoint!
- The adjoint of $\Sigma_F \circ \Delta_G$ is $\Pi_G \circ \Delta_F$.
- So what is the adjoint of the dual or the dual of the adjoint?
- Answer: $\Sigma_F \circ \Delta_G \;\mapsto\; \Pi_F \circ \Delta_G \;\mapsto\; \Sigma_G \circ \Delta_F$. (The other works too.)

On the level of spans, this is the transpose!

- The transpose operation is a composite of two more primitive ones.
- This doesn't happen within $\mathbb{S}$**pan**; kind of like contour integrals.

Similarly, the opposite of a category is a composite of two operations.

- A category $\mathcal{C}$ can be viewed as a monad in $\mathbb{S}$**pan**, and its adjoint is...
- ...a comonoid $c(1)y \triangleleft\!\xrightarrow{c}\!\triangleleft\; c(1)y$, which is a cat'y in a different way!
- And the dual of that comonoid is again a monad in $\mathbb{S}$**pan**, namely $\mathcal{C}^{\mathsf{op}}$.
- In particular, with $u = \sum_{N\in\mathbb{N}} y^N$, we will use (twice) that $\overline{\left[\begin{smallmatrix} u \\ u \end{smallmatrix}\right]} \simeq \mathbf{Fin}$.

Outline

Finitary instances

For $X\colon c \to \mathbf{Set}$, i.e. $c \triangleleft\!\!\overset{X}{\text{———}}\!\!\triangleleft\ 0$, the following are equivalent

- the copresheaf X is *finitary*, i.e. it factors through $\mathbf{Fin} \subseteq \mathbf{Set}$.
- there exists a function $\ulcorner X \urcorner\colon c(1) \to u(1)$ with

$$\begin{tikzcd} c(1)\mathcal{y} \arrow[d, "\ulcorner X\urcorner"'] \arrow[r, phantom, "\triangleleft\!\overset{X}{\text{———}}\!\triangleleft"] \arrow[dr, phantom, "\text{cart}"] & 0 \arrow[d, equal] \\ u(1)\mathcal{y} \arrow[r, phantom, "\triangleleft\!\underset{\overline{u}}{\text{———}}\!\triangleleft"] & 0 \end{tikzcd}$$

- There exists a monad map $\ulcorner X \urcorner_1$ as shown here:

$$\begin{tikzcd} c(1)\mathcal{y} \arrow[d, "\ulcorner X\urcorner"'] \arrow[r, phantom, "\triangleleft\!\overset{c^{\dagger}}{\text{————}}\!\triangleleft"] \arrow[dr, phantom, "\Downarrow\ulcorner X\urcorner_1"] & c(1)\mathcal{y} \arrow[d, "\ulcorner X\urcorner"] \\ u(1)\mathcal{y} \arrow[r, phantom, "\triangleleft\!\underset{\overline{\begin{bmatrix} u \\ u \end{bmatrix}}}{\text{————}}\!\triangleleft"] & u(1)\mathcal{y} \end{tikzcd}
\qquad c^{\dagger} = \sum_{a \in c(1)} c^{\mathrm{op}}[a]\mathcal{y}.$$

for which the $c^{\dagger}$-algebra induced by $\overline{u}$ is X.

Finitary instances

For $X\colon c \to \mathbf{Set}$, i.e. $c \xleftarrow{X} 0$, the following are equivalent

- the copresheaf X is *finitary*, i.e. it factors through $\mathbf{Fin} \subseteq \mathbf{Set}$.
- there exists a function $\ulcorner X \urcorner\colon c(1) \to u(1)$ with

$$\begin{tikzcd}
c(1)y \arrow[d, "\ulcorner X \urcorner"'] & 0 \arrow[l, "X"'] \arrow[d, equal] \arrow[dl, phantom, "\text{cart}"] \\
u(1)y & 0 \arrow[l, "\overline{u}"]
\end{tikzcd}$$

- There exists a monad map $\ulcorner X \urcorner_1$ as shown here:

$$\begin{tikzcd}
c(1)y \arrow[d, "\ulcorner X \urcorner"'] & c(1)y \arrow[l, "c^{\dagger}"'] \arrow[d, "\ulcorner X \urcorner"] \arrow[l, phantom, shift right=6, "\Downarrow \ulcorner X \urcorner_1"] \\
u(1)y & u(1)y \arrow[l, "\overline{\begin{bmatrix} u \\ u \end{bmatrix}}"]
\end{tikzcd}
\qquad c^{\dagger} = \sum_{a \in c(1)} c^{\mathrm{op}}[a]y.$$

  for which the $c^{\dagger}$-algebra induced by $\overline{u}$ is X.

I know that's impossible to follow; sorry! The gist: "everything works!"

Commutative monoids as Fin-algebras

A database schema assigns a comm've monoid $(M_a, \circledast_a)$ to each $a \in c(1)$.

- Assigning the set M_a to each $a \in c(1)$ is a bicomodule $y \xleftarrow{My} Ay$.
- Consider the following diagram that coerces $\circledast$ into the picture:

$$
\begin{tikzcd}
 & u(1)y & y \arrow[l, "u"'] \arrow[d, Rightarrow, "\circledast"'] & \\
u(1)y \arrow[ur, "\overline{\begin{bmatrix} u \\ u \end{bmatrix}}"] & & y \arrow[ll, "u"] & c(1)y \arrow[l, "My"] \arrow[ul, "My"']
\end{tikzcd}
$$

- The composite $u(1) \xleftarrow{u} y \xleftarrow{My} c(1)y$ assigns...
- ... to each $N \in \mathbb{N}$ and $a \in c(1)$ the set $(M_a)^N$.

Commutative monoids as Fin-algebras

A database schema assigns a comm've monoid $(M_a, \circledast_a)$ to each $a \in c(1)$.

- Assigning the set M_a to each $a \in c(1)$ is a bicomodule $y \overset{My}{\lhd\!\!-\!\!\!-\!\!\lhd} Ay$.
- Consider the following diagram that coerces $\circledast$ into the picture:

$$
\begin{tikzcd}
 & u(1)y \arrow[dl, "\overline{\begin{bmatrix} u \\ u \end{bmatrix}}"'] & y \arrow[l, "u"'] \arrow[d, phantom, "\circledast\Downarrow"] & \\
u(1)y & & y \arrow[ll, "u"] & c(1)y \arrow[l, "My"] \arrow[ul, "My"']
\end{tikzcd}
$$

- The composite $u(1) \overset{u}{\lhd\!\!-\!\!\!-\!\!\lhd} y \overset{My}{\lhd\!\!-\!\!\!-\!\!\lhd} c(1)y$ assigns...
- ... to each $N \in \mathbb{N}$ and $a \in c(1)$ the set $(M_a)^N$.

So what does the 2-cell say, and what does being a $\overline{\begin{bmatrix} u \\ u \end{bmatrix}}$-module mean?

- Given a function $f\colon N \to N'$, an object $a \in c(1)$, and $m \in (M_a)^N$...
- ...there is an induced $(\circledast m)_f \in (M_a)^{N'}$. Integration along fibers.
- $u \lhd My$ being a $\overline{\begin{bmatrix} u \\ u \end{bmatrix}}$-algebra means it works with ids and composites.

Aggregation

The thing we've worked so hard for is as follows.

- Suppose we have a category c and a copresheaf $X \colon c \to \mathbf{Set}$ and...
- ...a commutative monoid M_a and a map $s_a \colon X_a \to M_a$ for each $a \in c(1)$.
- Then given $f \colon a \to b$ in c, and given $y \in X(b)$, we want:
- ... to take the fiber $\{x \in X(a) \mid x.f = y\}$ and "add 'em up".
- That is, take $\circledast_{\{x \mid x.f = y\}} s_a(x)$.

$$\begin{tikzcd} X_a \arrow[r, "s_a"] \arrow[d, "X_f"'] & M_a \\ X_b \arrow[ur, dashed, "(\circledast\ s_a)_f"'] & \end{tikzcd}$$

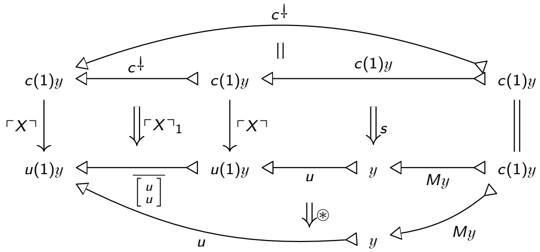
Aggregation

The thing we've worked so hard for is as follows.

- Suppose we have a category c and a copresheaf $X\colon c \to \mathbf{Set}$ and...
- ...a commutative monoid M_a and a map $s_a\colon X_a \to M_a$ for each $a \in c(1)$.
- Then given $f\colon a \to b$ in c, and given $y \in X(b)$, we want:
- ... to take the fiber $\{x \in X(a) \mid x.f = y\}$ and “add 'em up”.
- That is, take $\circledast_{\{x \mid x.f=y\}} s_a(x)$.

$$\begin{tikzcd} X_a \arrow[r, "s_a"] \arrow[d, "X_f"'] & M_a \\ X_b \arrow[ur, dashed, "(\circledast\, s_a)_f"'] & \end{tikzcd}$$

We have accomplished this now, using pieces we've collected.



Aggregation

The thing we've worked so hard for is as follows.

- Suppose we have a category c and a copresheaf $X\colon c \to \mathbf{Set}$ and...
- ...a commutative monoid M_a and a map $s_a\colon X_a \to M_a$ for each $a \in c(1)$.
- Then given $f\colon a \to b$ in c, and given $y \in X(b)$, we want:
- ... to take the fiber $\{x \in X(a) \mid x.f = y\}$ and “add 'em up”.
- That is, take $\circledast_{\{x \mid x.f=y\}} s_a(x)$.

$$\begin{tikzcd} X_a \arrow[r, "s_a"] \arrow[d, "X_f"'] & M_a \\ X_b \arrow[ur, dashed, "(\circledast\, s_a)_f"'] & \end{tikzcd}$$

We have accomplished this now, using pieces we've collected.

$$\begin{tikzcd}
b \arrow[d, "\ulcorner X \urcorner"'] & a \arrow[l, "f"'] \arrow[d, "\ulcorner X \urcorner"'] \arrow[l, phantom, shift right=6, "\Downarrow \ulcorner X \urcorner_1"] & & a \arrow[ll, equal] \arrow[lll, bend right, "f"'] \arrow[ll, phantom, shift left=4, "\|"] \arrow[d, equal] \\
u(1)y & u(1)y \arrow[l, "\left[\begin{smallmatrix} u \\ u \end{smallmatrix}\right]"] & y \arrow[l, "u"'] \arrow[u, phantom, "\Downarrow s"] & c(1)y \arrow[l, "My"'] \arrow[dl, bend left, "My"] \\
& & y \arrow[ull, bend left, "u"] \arrow[u, phantom, "\Downarrow \circledast"] &
\end{tikzcd}$$

Aggregation

The thing we've worked so hard for is as follows.

- Suppose we have a category c and a copresheaf $X : c \to \mathbf{Set}$ and...
- ...a commutative monoid M_a and a map $s_a : X_a \to M_a$ for each $a \in c(1)$.
- Then given $f : a \to b$ in c, and given $y \in X(b)$, we want:
- ... to take the fiber $\{x \in X(a) \mid x.f = y\}$ and "add 'em up".
- That is, take $\circledast_{\{x \mid x.f = y\}} s_a(x)$.

$$\begin{tikzcd} X_a \arrow[r, "s_a"] \arrow[d, "X_f"'] & M_a \\ X_b \arrow[ur, dashed, "(\circledast\ s_a)_f"'] & \end{tikzcd}$$

We have accomplished this now, using pieces we've collected.

$$\begin{tikzcd}[column sep=huge]
\color{purple} b \arrow[d, "\ulcorner X \urcorner"'] & \color{purple} a \arrow[l, "\color{purple} f"'] \arrow[r, equal] \arrow[d, "\ulcorner X \urcorner"] \arrow[dl, phantom, "\Downarrow \ulcorner X \urcorner_1"] \arrow[r, phantom, shift left=4, "\shortparallel"] & \color{purple} a \arrow[ll, bend right, "\color{purple} f"'] \arrow[d, equal] \arrow[dl, phantom, "\Downarrow s"] \\
\color{purple} X(b) & \color{purple} X(a) \arrow[l, "\color{purple} X(f)"] \arrow[r, dash, "\color{purple} x \mapsto s_a(x)"'] \arrow[dr, phantom, "\Downarrow \circledast"] & {} \\
 & & y \arrow[ull, bend left, "u"] \arrow[u, "My"']
\end{tikzcd}$$

Aggregation

The thing we've worked so hard for is as follows.

- Suppose we have a category c and a copresheaf $X\colon c \to \mathbf{Set}$ and...
- ...a commutative monoid M_a and a map $s_a\colon X_a \to M_a$ for each $a \in c(1)$.
- Then given $f\colon a \to b$ in c, and given $y \in X(b)$, we want:
- ... to take the fiber $\{x \in X(a) \mid x.f = y\}$ and "add 'em up".
- That is, take $\circledast_{\{x|x.f=y\}} s_a(x)$.

$$\begin{tikzcd}
X_a \arrow[r, "s_a"] \arrow[d, "X_f"'] & M_a \\
X_b \arrow[ur, dashed, "(\circledast\ s_a)_f"'] &
\end{tikzcd}$$

We have accomplished this now, using pieces we've collected.

$$\begin{tikzcd}[column sep=huge, row sep=large]
b \arrow[d, "\ulcorner X \urcorner"'] \arrow[dr, phantom, "\Downarrow \ulcorner X \urcorner_1"] & a \arrow[l, "f"'] \arrow[r, equal] \arrow[d, "\ulcorner X \urcorner"] \arrow[dr, phantom, "\Downarrow s"] & a \arrow[ll, bend right=30, "f"', ""{name=U, below}] \arrow[d, equal] \arrow[ll, phantom, bend right=15, "="] \\
X(b) & X(a) \arrow[l, "X(f)"] \arrow[r, dash, "x \mapsto s_a(x)"'] \arrow[d, phantom, "\Downarrow \circledast"] & {} \arrow[ll, dash, bend left=40, "y \mapsto \circledast_{\{x|x.f=y\}} s_a(x)"] \\
 & {} &
\end{tikzcd}$$

Outline

Summary

Aggregation is of central importance in database practice.

- Add up salaries, count things, collect each fiber into a set, etc.
- If we also have "calculated fields" (not too hard), you can...
- ... take averages, plot graphs, etc. Aggregation is very powerful.

Summary

Aggregation is of central importance in database practice.

- Add up salaries, count things, collect each fiber into a set, etc.
- If we also have "calculated fields" (not too hard), you can...
- ... take averages, plot graphs, etc. Aggregation is very powerful.

There's a really nice categorical story for data migration.

- It is that $\mathbb{C}\mathbf{at}^\sharp = \mathbb{C}\mathbf{omon}(\mathbf{Poly})$ is categories and prafunctors.
- And prafunctors are data migrations (e.g. find all paths in a graph).
- But a categorical formulation of aggregation has been missing.

Summary

Aggregation is of central importance in database practice.

- Add up salaries, count things, collect each fiber into a set, etc.
- If we also have "calculated fields" (not too hard), you can...
- ... take averages, plot graphs, etc. Aggregation is very powerful.

There's a really nice categorical story for data migration.

- It is that $\mathbb{C}\mathbf{at}^\sharp = \mathbb{C}\mathbf{omon}(\mathbf{Poly})$ is categories and prafunctors.
- And prafunctors are data migrations (e.g. find all paths in a graph).
- But a categorical formulation of aggregation has been missing.

But **Poly** is so highly-structured, we asked if it might include aggregation.

- Using adjoint prafunctors, local monoidal closures, and coclosures...
- ...we found a way to say what we needed to say.
- It's not as plain and simple as I'd like, but there's likely a better way.
- We tested the mettle of **Poly** and it was indeed up to the task!

Thank you for your time; questions and comments welcome!