

Functorial Aggregation

David I. Spivak



Workshop on Polynomial Functors
2022 March 18

Outline

1 Introduction

- Databases and aggregation
- Purity of methods
- Plan of the talk

2 Background on Poly

3 Cat#, home of data migration

4 Aggregation

5 Conclusion

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 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*?

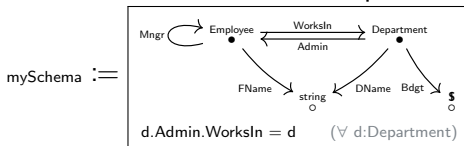
I think of mathematical fields as [accounting systems](#).

- 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 $F: \mathcal{C} \rightarrow \mathbf{Set}$.
- The site \mathcal{C} is called the *schema* and the copresheaf is the *instance*.



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

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

$$\begin{array}{ccc} Q & \longrightarrow & P \\ \downarrow & \lrcorner & \downarrow f \\ P & \xrightarrow{f} & B \end{array}$$

- 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 .

$$\begin{array}{ccc} P & \xrightarrow{s} & \mathbb{R} \\ f \downarrow & \nearrow & \\ B & & (\text{sum } s)_f \end{array}$$

- 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) \mathcal{C}, \mathcal{D} , a data migration functor...
- ... $\mathcal{C} \leftarrow \mathcal{D}$ is a parametric right adjoint $\mathcal{D}\text{-}\mathbf{Set} \rightarrow \mathcal{C}\text{-}\mathbf{Set}$.
- 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 $f' \rightarrow f$?

$$\begin{array}{ccccc}
 P' & \longrightarrow & P & \xrightarrow{s} & \mathbb{R} \\
 f' \downarrow & & f \downarrow & \nearrow & \uparrow \\
 B' & \longrightarrow & B & & (\text{sum } s)_f
 \end{array}$$

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{Cat}^\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 $\mathbf{Poly}(p, q \triangleleft r) \cong \mathbf{Poly}\left(\left[\begin{smallmatrix} r \\ p \end{smallmatrix} \right], q\right)$;
 - An indexed right \triangleleft -coclosure (Myers?), i.e. $\mathbf{Poly}(p, q \triangleleft r) \cong \sum_{f: p(1) \rightarrow q(1)} \mathbf{Poly}(p \overset{f}{\frown} q, r)$;
 - An indexed right \otimes -coclosure (Niu?), i.e. $\mathbf{Poly}(p, q \otimes r) \cong \sum_{f: p(1) \rightarrow q(1)} \mathbf{Poly}(p \overset{f}{\bowtie} q, r)$;
 - At least eight monoidal structures in total;
 - \triangleleft -monoids generalize plain operads;
 - \triangleleft -comonoids are exactly categories; bicomodules are data migrations. This is \mathbb{Cat}^\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 $E \rightarrow B$ and a map $E \rightarrow \mathbb{R}$, 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.
- Erdős 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 $\mathbb{Cat}^\sharp = \mathbb{Comon}(\mathbf{Poly})$, natural home of data migration.
- Show how aggregation-useful structures on **Poly** generalize to \mathbb{Cat}^\sharp .
- Explain aggregation with the structures we've explored.
- Conclude with a summary.

Outline

1 Introduction

2 Background on Poly

- **Poly**: polynomials in one variable
- Relevant categorical structures

3 \mathbf{Cat}^\sharp , home of data migration

4 Aggregation

5 Conclusion

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

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

- For any set E , denote the functor it represents by $y^E := \mathbf{Set}(E, -)$.
- 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:

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

Morphisms $p \xrightarrow{\varphi} q$ are just natural transformations $\mathbf{Set} \begin{array}{c} p \\ \Downarrow \varphi \\ q \end{array} \mathbf{Set}$

- Combinatorially, a map $\varphi: p \rightarrow q$ can be given in two parts:
- A function $\varphi_1: p(1) \rightarrow q(1)$ “forward on positions” and...
- ...for each $I \in p(1)$, a function $q[\varphi_1 I] \rightarrow p[I]$ “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, $\left(\sum_{I \in p(1)} p[I]\right) \rightarrow p(1) \dots$
- ...then $p \otimes q$ is just product of base and total spaces for $p, q \in \mathbf{Poly}$.

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

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

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

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

- The polynomial y is a *dualizing object*: for any $A \in \mathbf{Set} \dots$
- ...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: $(p_1 \triangleleft p_2) \otimes (q_1 \triangleleft q_2) \rightarrow (p_1 \otimes q_1) \triangleleft (p_2 \otimes q_2)$.

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

- We'll only need the former, $\mathbf{Poly}(p, q \triangleleft p') \cong \mathbf{Poly}\left(\left[\begin{smallmatrix} p' \\ p \end{smallmatrix}\right], q\right)$.
- For any $p \in \mathbf{Poly}$ the polynomial $\left[\begin{smallmatrix} p \\ p \end{smallmatrix}\right]$ 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}^{op} (Jacobs) spanned by p -fibers.

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

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

Outline

- 1 Introduction
- 2 Background on Poly
- 3 **Cat[#], home of data migration**
 - Shulman, Ahman-Uustalu, Garner
 - Databases for intuition
 - Categorical structures
- 4 Aggregation
- 5 Conclusion

$\mathbb{Cat}^\sharp := \mathbb{Comon}(\mathbf{Poly})$

Shulman: if equipment \mathbb{P} has good equalizers, $\mathbb{Comon}(\mathbb{P})$ 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 $\mathbb{Cat}^\sharp := \mathbb{Comon}(\mathbf{Poly})$ is also an equipment.

Ahman-Uustalu: the objects of $\mathbb{Comon}(\mathbf{Poly})$ are [categories](#).

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

Garner: horizontal morphisms of $\mathbb{Comon}(\mathbf{Poly})$ are [data migrations](#).

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

PolyFun and Span as subequipments of Cat[#]

To understand $\mathbf{Cat}^{\#} := \mathbf{Comon}(\mathbf{Poly})$, let's start with something familiar.

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

$$\begin{array}{ccccc}
 I & \longleftarrow & E & \longrightarrow & B \longrightarrow J \\
 \downarrow & & \varphi^{\#} \uparrow & & \parallel \\
 & & E' \times_{B'} B & \longrightarrow & B \\
 & & \downarrow \lrcorner & & \downarrow \varphi_1 \\
 I' & \longleftarrow & E' & \longrightarrow & B' \longrightarrow J'
 \end{array}
 \quad \rightsquigarrow \quad
 \begin{array}{ccc}
 Iy & \triangleleft^p & Jy \\
 \downarrow & \Downarrow \varphi & \downarrow \\
 I'y & \triangleleft_{p'} & J'y
 \end{array}$$

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

The usual double cat'y of spans sits inside: $\mathbf{Span} \subseteq \mathbf{PolyFun} \subseteq \mathbf{Cat}^{\#}$.

- $\mathbf{Span} \subseteq \mathbf{Cat}^{\#}$ 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 $\mathbb{C}at^\sharp$ by thinking about databases.

- The indexing cat'y for graphs is $\mathcal{G} := \boxed{\bullet \rightrightarrows \bullet}$, carried by $g := y^3 + y$.
- Bicomodules $c \leftarrow^X \triangleleft 0$ can be ident'd with copresheaves $c \xrightarrow{X} \mathbf{Set}$.
- So a graph is just a bicomodule $g \leftarrow^X \triangleleft 0$. Call X a g -set or \mathcal{G} -set.
- Think of \mathcal{G} as a database *schema*, an arrangement for sets, and...
- ...think of $X: \mathcal{G} \rightarrow \mathbf{Set}$ 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 $\mathcal{C}\text{-}\mathbf{Set}(Q, -): \mathcal{C}\text{-}\mathbf{Set} \rightarrow \mathbf{Set}$.
- 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) $\mathcal{C} \leftarrow^P \triangleleft \mathcal{D}$ 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 $\mathcal{D} \leftarrow \triangleleft 0$, composition returns a \mathcal{C} -instance.

Adjoint prafunctors

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

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

When is a prafunctor $c \xleftarrow{p} \triangleleft d$ a right adjoint in \mathbb{Cat}^\sharp ?

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

In the subcategory $\mathbb{PolyFun} = \mathbb{Cat}_{\text{disc}}^\sharp$, 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 $\mathbb{C}at^\sharp$ has lots of structure, e.g. it is monoidal.

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

It also has local \otimes -monoidal structures.

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

$$\begin{array}{ccc}
 c_0 \triangleleft \xrightarrow[p_1]{q_1} c_1 \triangleleft \xrightarrow[p_2]{q_2} c_2 & \rightsquigarrow & c_0 \triangleleft \xrightarrow[(p_1 c_0 \otimes_{c_1} q_1) \triangleleft_{c_1} (p_2 c_1 \otimes_{c_2} q_2)]{(p_1 \triangleleft_{c_1} p_2) c_0 \otimes_{c_2} (q_1 \triangleleft_{c_1} q_2)} c_2 \\
 & & \Downarrow \text{duoid}
 \end{array}$$

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

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

- 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. $Cy \xleftarrow{CDy} Dy$.
- Calling it $\perp := CDy$, the functor $\bar{\cdot} := {}_c[-, \perp]_D$ provides a duality...
- If $p \in C\text{-}\mathbf{Set}[D]$ is linear then $[p, \perp]$ is conjunctive, and vice versa.
- In particular $\bar{\bar{p}} \cong p$ for any linear or conjunctive p .
- It generalizes $[Ay, y] \cong y^A$ from earlier.

Transposing a span, “oppositing” a category

The idea of $\bar{\rho}$ 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 **Span**; 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 **Span**, and its adjoint is...
- ...a comonoid $c(1)y \xleftarrow{c} \triangleleft c(1)y$, which is a cat'y in a different way!
- And the dual of that comonoid is again a monad in **Span**, namely \mathcal{C}^{op} .
- In particular, with $u = \sum_{N \in \mathbb{N}} y^N$, we will use (twice) that $\overline{\begin{bmatrix} u \\ u \end{bmatrix}} \simeq \mathbf{Fin}$.

Outline

- 1 Introduction
- 2 Background on Poly
- 3 Cat^\sharp , home of data migration
- 4 **Aggregation**
 - Finitary instances
 - Commutative monoids
 - Putting it together
- 5 Conclusion

Finitary instances

For $X: c \rightarrow \mathbf{Set}$, i.e. $c \triangleleft \xrightarrow{X} \triangleleft 0$, the following are equivalent

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

$$\begin{array}{ccc} c(1)y & \triangleleft \xrightarrow{X} \triangleleft & 0 \\ \lceil X \rceil \downarrow & \text{cart} & \parallel \\ u(1)y & \triangleleft \xrightarrow{\bar{u}} \triangleleft & 0 \end{array}$$

- There exists a monad map $\lceil X \rceil_1$ as shown here:

$$\begin{array}{ccc} c(1)y & \triangleleft \xrightarrow{c^\dagger} \triangleleft & c(1)y \\ \lceil X \rceil \downarrow & \Downarrow \lceil X \rceil_1 & \downarrow \lceil X \rceil \\ u(1)y & \triangleleft \xrightarrow{\begin{bmatrix} u \\ u \end{bmatrix}} \triangleleft & u(1)y \end{array} \quad c^\dagger = \sum_{a \in c(1)} c^{\text{op}}[a]y.$$

for which the c^\dagger -algebra induced by \bar{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'ive monoid (M_a, \otimes_a) to each $a \in c(1)$.

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

$$\begin{array}{ccccc}
 & & u(1)y & \xleftarrow{u} & y \\
 & \nearrow \overline{\begin{bmatrix} u \\ u \end{bmatrix}} & \downarrow \otimes & & \nwarrow M_y \\
 u(1)y & \xleftarrow{u} & y & \xleftarrow{M_y} & c(1)y
 \end{array}$$

- The composite $u(1) \xleftarrow{u} y \xleftarrow{M_y} c(1)y$ 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 $f: N \rightarrow N'$, an object $a \in c(1)$, and $m \in (M_a)^N$...
- ...there is an induced $(\otimes m)_f \in (M_a)^{N'}$. [Integration along fibers](#).
- $u \triangleleft M_y$ 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 $X: c \rightarrow \mathbf{Set}$ and...
- ...a commutative monoid M_a and a map $s_a: X_a \rightarrow M_a$ for each $a \in c(1)$.
- Then given $f: a \rightarrow b$ 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 $\bigoplus_{\{x \mid x.f=y\}} s_b(x)$.

$$\begin{array}{ccc}
 X_a & \xrightarrow{s_a} & M_a \\
 X_f \downarrow & \nearrow (\otimes s)_f & \\
 X_b & &
 \end{array}
 \quad \dots \text{or better...} \quad
 \begin{array}{ccc}
 X_b & \xleftarrow{X_f} & X_a \\
 (\otimes s)_f \searrow & & \downarrow s_a \\
 & & M_a
 \end{array}$$

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

$$\begin{array}{ccccccc}
 & & c^\dagger & & & & \\
 & \swarrow & \text{---} & \searrow & \text{---} & \swarrow & \\
 c(1)y & & c(1)y & & c(1)y & & c(1)y \\
 \downarrow \lceil X^\top & \Downarrow \lceil X^\top_1 & \downarrow \lceil X^\top & \Downarrow s & \Downarrow & & \\
 u(1)y & \xleftarrow{\quad} & u(1)y & \xleftarrow{u} & y & \xleftarrow{M_y} & c(1)y \\
 & \nwarrow \left[\begin{smallmatrix} u \\ u \end{smallmatrix} \right] & & \downarrow \otimes & \nwarrow M_y & & \\
 & & & y & & &
 \end{array}$$

u

Outline

- 1 Introduction
- 2 Background on Poly
- 3 Cat#, home of data migration
- 4 Aggregation
- 5 Conclusion**
 - Summary

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 $\mathbb{Cat}^\sharp = \mathbb{Comon}(\mathbf{Poly})$ 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!