ACT 2024 - Proqueries and Praqueries

Gabriel Goren Roig [1 2], Joshua Meyers [3], Emilio Minichiello [3], Ryan Wisnesky [3]

June 17, 2024

[1]Departamento de Matemática, Universidad de Buenos Aires, Argentina
[2]Instituto de Ciencias de la Computación (ICC), CONICET

[3]Conexus AI

Introduction

- Going to talk about some WIP in categorical database theory.
- This work expands the algebraic model of categorical database theory developed in “Algebraic Databases” [Sch+17] and “Algebraic Data Integration” [SW17].
- (Some of) the main results:
  - Introduce (non-strict) proqueries, data transformations similar to **conjunctive queries**,
  - Prove correctness of proquery presentation composition algorithm (building off of a similar algorithm for uberflower composition in [SW17]),
  - Introduce praqueries, data transformations similar to **unions of conjunctive queries**,
  - Introduce and prove correctness of praquery presentation composition algorithm.

Categorical Database Theory: the story so far

- 1970 - Relational database theory is born [Cod70]
- Beginning of overlap between DB and CT[1] - [BS81], [LS90], [RW91], [JD94]
- Sketch Data Model (Rosebrugh, Johnson) - [JRW00], [JR02]
- Modern Iteration -
  - Spivak (2012) - Functorial Data Migration [Spi12]
  - Spivak, Wisnesky (2015) - Relational Foundations for Functorial Data Migration [SW15]
  - Schultz, Wisnesky (2017) - Algebraic Data Integration [SW17]
  - Schultz, Vasilakopoulou, Wisnesky, Spivak (2017) - Algebraic Databases [Sch+17]
  - Lynch, Patterson, Fairbanks - Categorical data structures for technical computing [PLF22]

[1] References from Rosebrugh's talk [Ros]

Categorical Database Theory: the story so far

- 1970 - Relational database theory is born [Cod70]
- Beginning of overlap between DB and CT[2] - [BS81], [LS90], [RW91], [JD94]
- Sketch Data Model (Rosebrugh, Johnson) - [JRW00], [JR02]
- Modern Iteration -
  - Spivak (2012) - Functorial Data Migration [Spi12]
  - Spivak, Wisnesky (2015) - Relational Foundations for Functorial Data Migration [SW15]
  - Schultz, Wisnesky (2017) - Algebraic Data Integration [SW17]
  - Schultz, Vasilakopoulou, Wisnesky, Spivak (2017) - Algebraic Databases [Sch+17]
  - Lynch, Patterson, Fairbanks - Categorical data structures for technical computing [PLF22]

[2] References from Rosebrugh's talk [Ros]

The Functorial Data Model

The Functorial Data Model

Let us quickly recall the Functorial Data Model [Spi12].

$$\text{Database Schema} \longleftrightarrow \text{(small) Category } \mathcal{C}$$

$$\text{Database Instance} \longleftrightarrow \text{Copresheaf } \mathcal{I} : \mathcal{C} \to \mathbf{Set}$$

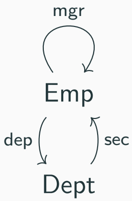
The Functorial Data Model

Really what we are interested in are **category presentations**.

A cat pres. C consists of sets

$$\mathrm{Sort}(C), \quad \mathrm{Fun}(C), \quad \text{and} \quad \mathrm{Eq}(C).$$

Example:



$$\mathrm{mgr.dep} = \mathrm{dep}$$

$$\mathrm{mgr.mgr} = \mathrm{mgr}$$

$$\mathrm{sec.dep} = 1_{\mathrm{Dept}}$$

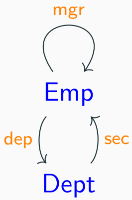
The Functorial Data Model

Really what we are interested in are **category presentations**.

A cat pres. C consists of sets

$$\text{Sort}(C), \quad \text{Fun}(C), \quad \text{and} \quad \text{Eq}(C).$$

Example:



$$\text{mgr.dep} = \text{dep}$$

$$\text{mgr.mgr} = \text{mgr}$$

$$\text{sec.dep} = 1_{\text{Dept}}$$

Note: Equations are between paths, written $p =_C q$ and composition is written left to right.

The Functorial Data Model

Given a category presentation C, we let $(\!|C|\!)$ denote the category it presents. We also call $(\!|C|\!)$ the **semantics** of C. Its objects are the sorts of C and its morphisms are the paths in C, modulo equations.

More formally $(\!|C|\!)(c, c')$ is the set of paths from c to c' modulo the **provable equality relation** $\approx_C$, defined as follows:

$$\frac{p =_C q}{p \approx_C q} \qquad \frac{}{p \approx_C p} \qquad \frac{p \approx_C q}{q \approx_C p} \qquad \frac{p \approx_C q \qquad q \approx_C r}{p \approx_C r}$$

$$\frac{f : c \to c' \qquad p \approx_C q : c' \to c''}{f.p \approx_C f.q}$$

$$\frac{f : c' \to c'' \qquad p \approx_C q : c \to c'}{p.f \approx_C q.f}$$

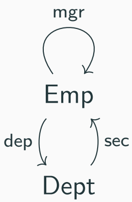
The Functorial Data Model

We have the following sets of morphisms

$$(\!|C|\!)(\mathsf{Emp}, \mathsf{Dept}) = \{[\mathsf{dep}] = [\mathsf{dep.sec.dep}] = [\mathsf{mgr.dep}] = [\mathsf{mgr.mgr.dep}]\},$$

$$(\!|C|\!)(\mathsf{Dept}, \mathsf{Emp}) = \{[\mathsf{sec}] = [\mathsf{sec.dep.sec}], [\mathsf{sec.mgr}]\}$$

Example:



$$\mathsf{mgr.dep} = \mathsf{dep}$$

$$\mathsf{mgr.mgr} = \mathsf{mgr}$$

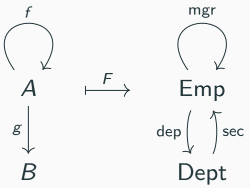
$$\mathsf{sec.dep} = 1_{\mathsf{Dept}}$$

The Functorial Data Model

Can define schema/category presentation morphisms $F : C \to D$ by functions $F_0 : \mathsf{Sort}(C) \to \mathsf{Sort}(D)$ and $F_1 : \mathsf{Fun}(C) \to \mathsf{Path}(D)$. Let F denote the extension of F_1 to paths.

We require that if $p =_C q$, then $F(p) \approx_C F(q)$.

$f.g = f.f.g$



$\mathsf{mgr.dep} = \mathsf{dep}$

$\mathsf{mgr.mgr} = \mathsf{mgr}$

$\mathsf{sec.dep} = 1_{\mathsf{Dept}}$

$$F(f.g) \neq F(f.f.g), \quad \text{but} \quad F(f.g) \approx_C F(f.f.g)$$

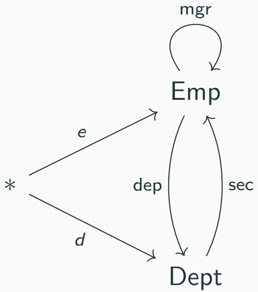
$$\mathsf{mgr.dep} \neq \mathsf{mgr.mgr.dep}, \quad \text{but} \quad \mathsf{mgr.dep} \approx_C \mathsf{mgr.mgr.dep}$$

The Functorial Data Model

Get a category **CatPr** with semantics functor

$$(\!|-|\!) : \mathbf{CatPr} \to \mathbf{Cat}$$

Can also define **instance presentations**.



$$\text{mgr.dep} = \text{dep}$$

$$\text{mgr.mgr} = \text{mgr}$$

$$\text{sec.dep} = 1_{\text{Dept}}$$

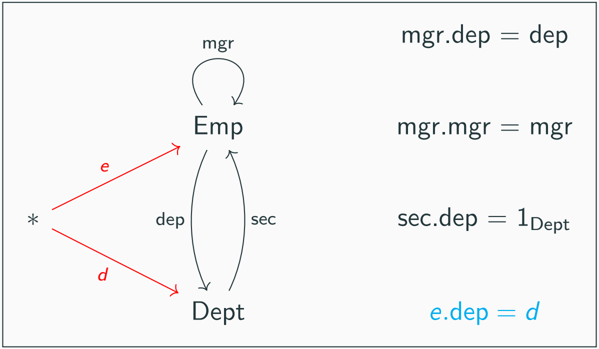
$$e.\text{dep} = d$$

The Functorial Data Model

An instance presentation I consists of a collection of **generators** and **equations**. We write $I = \langle I_\Gamma \mid I_E \rangle$.

Example:

$$I = \langle e : \text{Emp}, d : \text{Dept} \mid e.\text{dep} = d \rangle$$



The Functorial Data Model

Given an instance presentation

$$I = \langle e : \text{Emp}, d : \text{Dept} \mid e.\text{dep} = d \rangle$$

we can display it using tables:

Emp	mgr	dep
e	e.mgr	d
e.mgr	e.mgr	d
d.sec	d.sec.mgr	d
d.sec.mgr	d.sec.mgr	d

Dept	sec
d	d.sec

These are analogous to relational tables of **incomplete information** [Are+14, Section 2.3], also called **labelled nulls**.

The Functorial Data Model

Given an instance presentation I on a schema presentation C, we obtain semantics $[\![I]\!] : (\!|C|\!) \to \mathbf{Set}$ by setting

$$[\![I]\!](c) = \{* \to c\}/\approx_{\mathsf{Eq}(C)\cup I_E}$$

Morphisms of instance presentations $\varphi : I \to J$ over a schema presentation C require $*$ to be sent to $*$ and are the identity on C.

Get category $C\mathbf{InstPr}$ and semantics functor

$$[\![-]\!] : C\mathbf{InstPr} \to \mathbf{Set}^{(\!|C|\!)}$$

The Functorial Data Model

Note however, that there is no actual “data” in our tables. We merely keep track of **primary keys** and **foreign keys**.

Emp	mgr	dep
e	e.mgr	d
e.mgr	e.mgr	d
d.sec	d.sec.mgr	d
d.sec.mgr	d.sec.mgr	d

Dept	sec
d	d.sec

To add in “data”, we use the **Algebraic Model** [Sch+17], [SW17].

The Algebraic Data Model

The Algebraic Data Model

An **algebraic signature** Σ consists of sets:

$$\mathrm{Sort}(\Sigma), \qquad \text{and} \qquad \mathrm{Fun}(\Sigma),$$

But now, function symbols are allowed to have higher **arity**.

We say $f : (s_1, \ldots, s_n) \to s$ has arity n.

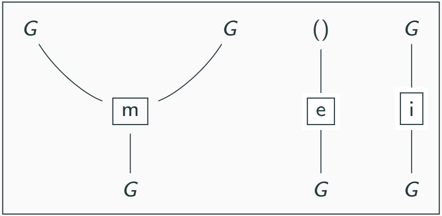
We call 0-ary function symbols **constant symbols**.

Example: Algebraic signature for groups Σ_{Grp}

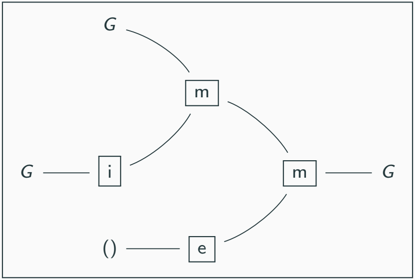
$$\mathrm{Sort}(\Sigma_{\mathrm{Grp}}) = \{G\},$$
$$\mathrm{Fun}(\Sigma_{\mathrm{Grp}}) = \{m : (G, G) \to G, e : () \to G, i : G \to G\}$$

The Algebraic Data Model

We can visualize function symbols as follows

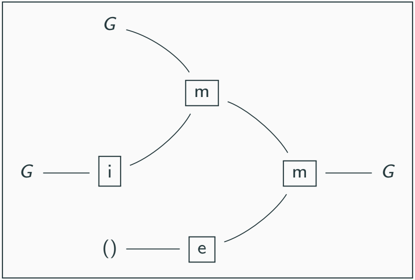


We can build **terms** by putting these function symbols together



The Algebraic Data Model

We use a convenient notation for these terms. For example:



Can be written as

$$x, y : G \vdash m(m(x, i(y)), e) : G$$

The Algebraic Data Model

Use the terminology:

$$\overbrace{\overbrace{x, y : G}^{\text{Context}} \vdash m(m(x, i(y)), e) : \overbrace{G}^{\text{Sort}}}^{\text{Term}}$$

An **algebraic presentation** T consists of an algebraic signature $(\mathrm{Sort}(T), \mathrm{Fun}(T))$ and a set $\mathrm{Eq}(T)$ of **equtions** between terms

Example: Algebraic presentation of groups T_{Grp}

$$\begin{aligned}
\mathrm{Sort}(T_{\mathrm{Grp}}) =& \{G\}, \\
\mathrm{Fun}(T_{\mathrm{Grp}}) =& \{m : (G, G) \to G, e : () \to G, i : G \to G\} \\
\mathrm{Eq}(T_{\mathrm{Grp}}) =& \{[x, y, z : G \vdash m(m(x, y), z) = m(x, m(y, z)) : G]\,, \\
& [x : G \vdash m(x, e) = x]\,, [x : G \vdash m(e, x) = x] \\
& [x : G \vdash m(x, i(x)) = e]\,, [x : G \vdash m(i(x), x) = e]\}
\end{aligned}$$

The Algebraic Data Model

If T is an algebraic presentation, we can define an equivalence relation $\approx_T$ on terms, similar to category presentations.

Let $[\![T]\!]$ denote the category whose objects are contexts, and morphisms are terms modulo $\approx_T$.

The function symbols produce morphisms

$$G \times G \xrightarrow{m} G, \qquad * \xrightarrow{e} G, \qquad G \xrightarrow{i} G$$

Morphisms of algebraic presentations $F : T \to T'$

$$\begin{aligned} \text{Sorts } s &\mapsto \text{Sorts } F(s) \\ \text{Function Symbols } f &\mapsto \text{Terms } F(f) \\ \text{Equations } t =_T t' &\mapsto \text{Provable Equations } F(t) \approx_{T'} F(t') \end{aligned}$$

Semantics gives a functor $[\![-]\!] : \mathbf{AlgPr} \to \mathbf{FPCat}$.

The Algebraic Data Model

Let us fix an algebraic presentation Ty, that we call the **typeside**.

Example:

$$\mathrm{Sort}(\mathrm{Ty}) = \{\mathrm{Str}, \mathrm{Int}\},$$

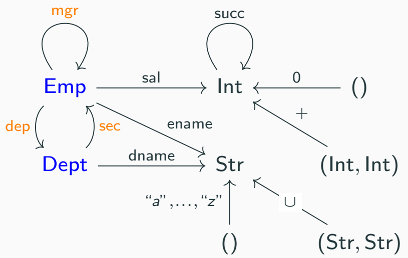
$$\begin{aligned}\mathrm{Fun}(\mathrm{Ty}) = \{&+ : (\mathrm{Int}, \mathrm{Int}) \to \mathrm{Int}, \mathrm{succ} : \mathrm{Int} \to \mathrm{Int}, 0 : () \to \mathrm{Int}, \\ &\cup : (\mathrm{Str}, \mathrm{Str}) \to \mathrm{Str}, \text{“}a\text{”}, \ldots, \text{“}z\text{”} : () \to \mathrm{Str}\}\end{aligned}$$

Now let us define an **algebraic schema presentation**.

The Algebraic Data Model

An algebraic schema presentation U over a fixed typeside Ty, consists of

- Entityside: Entities, Foreign Keys, Entity Equations,
- Typeside,
- Attributes, Schema Equations



$e : \text{Emp} \vdash e.\text{mgr}.\text{dep} = e.\text{dep} : \text{Dept}$

$e : \text{Emp} \vdash e.\text{mgr}.\text{mgr} = e.\text{mgr} : \text{Emp}$

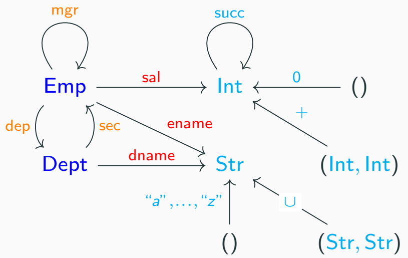
$d : \text{Dept} \vdash d.\text{sec}.\text{dep} = d : \text{Dept}$

$e : \text{Emp} \vdash e.\text{mgr}.\text{sal} = e.\text{sal} + 100 : \text{Int}$

The Algebraic Data Model

An algebraic schema presentation U over a fixed typeside Ty, consists of

- Entityside: Entities, Foreign Keys, Entity Equations,
- Typeside,
- Attributes, Schema Equations



$e : \mathsf{Emp} \vdash e.\mathsf{mgr}.\mathsf{dep} = e.\mathsf{dep} : \mathsf{Dept}$

$e : \mathsf{Emp} \vdash e.\mathsf{mgr}.\mathsf{mgr} = e.\mathsf{mgr} : \mathsf{Emp}$

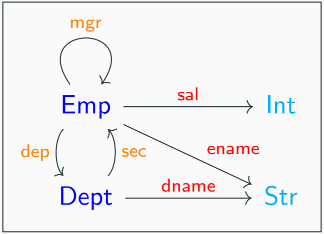
$d : \mathsf{Dept} \vdash d.\mathsf{sec}.\mathsf{dep} = d : \mathsf{Dept}$

$e : \mathsf{Emp} \vdash e.\mathsf{mgr}.\mathsf{sal} = e.\mathsf{sal} + 100 : \mathsf{Int}$

Note: We typically do not denote the typeside operations. They are understood from the definition of Ty.

The Algebraic Data Model

So an algebraic schema presentation looks like the following:



We want semantics to reflect this structure.

The Algebraic Data Model

Def: Given categories $\mathcal{C}, \mathcal{D}$, a bipartite category[3] $\mathcal{E} : \mathcal{C} \nrightarrow \mathcal{D}$ consists of a category $\mathcal{E}$ equipped with a functor $\pi : \mathcal{E} \to \mathbf{2}$ such that $\pi^{-1}(0) = \mathcal{C}$ and $\pi^{-1}(1) = \mathcal{D}$.

Thus if U is a schema presentation, then we obtain a bipartite category

$$(\!|U|\!) : \overbrace{(\!|U_e|\!)}^{\text{Entity category}} \nrightarrow \overbrace{[\![\mathrm{Ty}]\!]}^{\text{Typeside}}$$

This inspires the following definition.

[3]Equivalently a profunctor

The Algebraic Data Model

Def: Given a fixed finite product category $\mathcal{T}$, a **schema** consists of a bipartite category

$$\mathcal{U} : \mathcal{U}_e \nrightarrow \mathcal{T},$$

such that the inclusion $\mathcal{T} \hookrightarrow \mathcal{U}$ preserves finite products[4].

An **instance** on $\mathcal{U}$ is a functor $\mathcal{I} : \mathcal{U} \to \mathbf{Set}$ that preserves the finite products of $\mathcal{T}$.

We can define schema presentations and instance presentations as before.

This allows us to input schemas and presentations into a computer.

[4] Also called an algebraic profunctor in [Sch+17]

The Algebraic Data Model

In the algebraic model, instances can now display data other than labelled nulls, while still having incomplete information.

Example:

$$I_\Gamma = (e_0, e_1, e_2 : \mathrm{Emp}, d_0, d_1 : \mathrm{Dept})$$

$$I_E = \left\{ \begin{array}{l} e_0.\mathrm{ename} = \text{“Alice”},\ e_1.\mathrm{ename} = \text{“Bob”},\ e_2.\mathrm{ename} = \text{“Charlie”}, \\ d_0.\mathrm{dname} = \text{“CS”}, \qquad d_1.\mathrm{dname} = \text{“Math”}, \\ e_0.\mathrm{mgr} = e_0, \qquad e_1.\mathrm{mgr} = e_2, \qquad e_2.\mathrm{mgr} = e_2, \\ \qquad\qquad \ldots \end{array} \right\}$$

Emp	mgr	dep	sal	ename
e_0	e_0	d_0	100	“Alice”
e_1	e_2	d_1	e_1.sal	“Bob”
e_2	e_2	d_1	e_2.sal	“Charlie”

Dept	sec	dname
d_0	e_0	“CS”
d_1	e_2	“Math”

The Algebraic Data Model

Now in order to really call ourselves database theorists, we need to be able to **query** our data.

Idea of Proqueries: Profunctors between algebraic schemas.

Def: Given a schema $\mathcal{U}$ and $u \in \mathcal{U}$, let $y(u)$ denote the $\mathcal{U}$-instance given by

$$y(u)(u') = \mathcal{U}(u, u').$$

Call this the **representable instance** on u.

Can be presented very easily:

$$y(u) = [\![\langle \overbrace{x : u}^{\text{1 generator}} \mid \overbrace{\varnothing}^{\text{no equations}} \rangle]\!]$$

The Algebraic Data Model

Def: Given schemas $\mathcal{U}$ and $\mathcal{V}$, a (strict) **proquery** is a functor $\mathcal{P} : \mathcal{U}^{\mathsf{op}} \to \mathcal{V}\mathbf{Inst}$, such that $\mathcal{P}(t) = y(t)$, for every $t \in \mathsf{Ty}$, where $y(t)$ is the representable instance on t.

In general, we allow proqueries to have $\mathcal{P}(t) \cong y(t)$. However, every proquery is isomorphic to a strict proquery.

Think of analogue of profunctor as $\mathcal{P} : \mathcal{C}^{\mathsf{op}} \to \mathbf{Set}^{\mathcal{D}}$ with an extra twist due to attributes.

The Algebraic Data Model

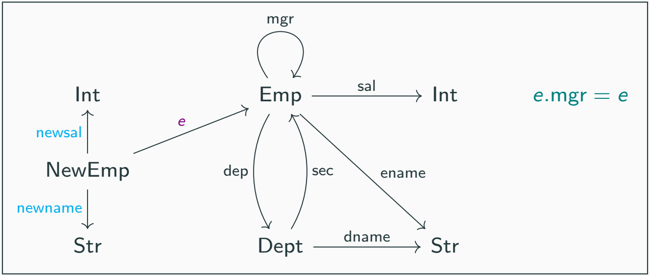
Easier to understand using presentations.

Proquery P : "Select the name and (salary + 50) of all employees who are their own manager"

SELECT e.ename AS newname, e.sal + 50 AS newsal,

FROM e : Emp,

WHERE e.mgr = e;



The Algebraic Data Model

SELECT e.ename AS newname, e.sal + 50 AS newsal,

FROM e : Emp,

WHERE e.mgr = e;

This gives a diagram of *U*-instance presentations.

$$y(\mathrm{Int}) \xrightarrow{\text{newsal}} P(\mathrm{NewEmp}) \xleftarrow{\text{newname}} y(\mathrm{Str})$$

$$\langle n : \mathrm{Int} \mid \varnothing \rangle \xrightarrow{\text{newsal}} \langle e : \mathrm{Emp} \mid e.\mathrm{mgr} = e \rangle \xleftarrow{\text{newname}} \langle s : \mathrm{Str} \mid \varnothing \rangle$$

$$\text{newsal} = (n \mapsto [e : \mathrm{Emp} \vdash e.\mathrm{sal} + 50 : \mathrm{Int}])$$
$$\text{newname} = (s \mapsto [e : \mathrm{Emp} \vdash e.\mathrm{ename} : \mathrm{Str}])$$

Given a proquery $\mathscr{P} : \mathcal{U} \nrightarrow \mathcal{V}$, get a functor

$$\Gamma_{\mathscr{P}} : \mathcal{V}\mathbf{Inst} \to \mathcal{U}\mathbf{Inst}$$

This is called the **evaluation** functor. Defined for $\mathcal{I} \in \mathcal{V}\mathbf{Inst}$ and $u \in \widetilde{\mathcal{U}}$ by

$$\Gamma_{\mathscr{P}}(\mathcal{I})(u) = \mathcal{V}\mathbf{Inst}(\mathscr{P}(u), \mathcal{I})$$

It has a left adjoint Λ, called **co-evaluation**.

Thm[Sch+17, Thm 8.10] If $F : \mathcal{V}\mathbf{Inst} \to \mathcal{U}\mathbf{Inst}$ is a functor such that $F(\mathcal{I})(t) \cong \mathcal{I}(t)$ and is a right adjoint, then there exists a proquery $\mathscr{P}$ such that $F \cong \Gamma_{\mathscr{P}}$.

The Algebraic Data Model

Proquery $P : U \nrightarrow V$

SELECT e.ename AS newname, e.sal + 50 AS newsal,

FROM e : Emp,

WHERE e.mgr = e;

V-Instance I

Emp	mgr	dep	sal	ename
e_0	e_0	d_0	100	"Alice"
e_1	e_2	d_1	e_1.sal	"Bob"
e_2	e_2	d_1	e_2.sal	"Charlie"

Dept	sec	dname
d_0	e_0	"CS"
d_1	e_2	"Math"

U-Instance $\Gamma_P(I)$

NewEmp	newname	newsal
e_0	"Alice"	150
e_2	"Charlie"	e_2.sal + 50

New Work

New Work

Given proqueries $\mathscr{P} : \mathscr{U} \nrightarrow \mathscr{V}$ and $\mathscr{Q} : \mathscr{V} \nrightarrow \mathscr{W}$, we can compose them by setting

$$(\mathscr{P} \odot \mathscr{Q})(u) = \int^{v \in \mathscr{V}} \mathscr{P}(u, v) \cdot \mathscr{Q}(v)$$

taken in the category $\mathscr{W}$**Inst**. Analogous to **subquery unnesting** or **view unfolding** in database theory.

However, in [Sch+17] and [SW17], two **inequivalent** notions of proquery presentation are given.

- Called **bimodule presentations** in [Sch+17], and
- Called **uberflowers** in [SW17].

A composition operation for uberflowers is sketched, but never proven to be semantically correct.

A composition operation for bimodule presentations is not given.

New Work

It turns out that a “semantically correct” and finite-preserving composition operation cannot be given for bimodule presentations!

This motivated us to write “Presenting Profunctors” [RMM24].

New Contribution: Give fully specified definition of proquery presentation and prove their correctness, i.e. define a composition operation $P \circledast Q$ such that $(\!| P \circledast Q |\!) \cong (\!| P |\!) \odot (\!| Q |\!)$, and this preserves **finiteness** of the presentations.

New Work

New Contribution: We introduce **praqueries**. These are similar to proqueries using the following analogy:

proqueries $\sim$ conjunctive queries

praqueries $\sim$ unions of conjunctive queries

Def: Given schemas $\mathcal{U}$ and $\mathcal{V}$, a praquery $\mathscr{P} : \mathcal{U} \nrightarrow \mathcal{V}$ consists of

- an instance $\mathscr{P}_0 : \mathcal{U}\textbf{Inst}$ such that $\mathscr{P}_0(t) = *$ for all $t \in \text{Ty}$,
- a proquery $\mathscr{P}_1 : \int \mathscr{P}_0 \nrightarrow \mathcal{V}$.

Given a praquery $\mathscr{P}$, get an evaluation functor $\Gamma_{\mathscr{P}} : \mathcal{V}\textbf{Inst} \to \mathcal{U}\textbf{Inst}$ by

$$\Gamma_{\mathscr{P}}(\mathcal{I})(u) = \sum_{x \in \mathscr{P}_0(u)} \mathcal{V}\textbf{Inst}(\mathscr{P}_1(x), \mathcal{I}).$$

New Work

Praquery P : "Select the name and (salary + 50) of all employees who are their own manager OR the name and salary of all employees in the Math department"

```
SELECT e.ename AS newname, e.sal + 50 AS newsal,
FROM e : Emp,
WHERE e.mgr = e;
UNION
SELECT e'.ename AS newname, e'.sal AS newsal,
FROM e' : Emp,
WHERE e'.dep.dname = "Math";
```

New Work

Praquery P : "Select the name and (salary + 50) of all employees who are their own manager OR the name and salary of all employees in the Math department"

V-Instance I

Emp	mgr	dep	sal	ename
e_0	e_0	d_0	100	"Alice"
e_1	e_2	d_1	50	"Bob"
e_2	e_2	d_1	100	"Charlie"

Dept	sec	dname
d_0	e_0	"CS"
d_1	e_2	"Math"

U-Instance $\mathrm{Eval}_P(I)$

NewEmp	newname	newsal
e_0	"Alice"	150
e_2	"Charlie"	150
e'_1	"Bob"	50
e'_2	"Charlie"	100

New Work

In our new work we:

- Give a definition of praquery presentation, their semantics and a composition operation.
- Prove correctness of composition of praquery presentations.
- Prove that praqueries can equivalently be described by those functors $\mathscr{P} : \mathscr{V}\mathbf{Inst} \to \mathscr{U}\mathbf{Inst}$ that preserve type-algebras and are **parametric right adjoint**/**prafunctors**.

Def: A functor $F : \mathscr{C} \to \mathscr{D}$ where $\mathscr{C}$ has a terminal object 1 is called a **parametric right adjoint** if in the factorization

$$\mathscr{C} \xrightarrow{F_1} \mathscr{D}/F(1) \xrightarrow{\Sigma} \mathscr{D}$$

the functor F_1 has a right adjoint.

These kinds of functors have very interesting properties and show up in many places in category theory: [Sha21], [GK12], [NS23].

New Work

Thank you!

Questions? Comments? Email me at eminichiello67@gmail.com

Check out implementation of this math using the CQL language at https://www.categoricaldata.net/

References

[Are+14] Marcelo Arenas et al. **Foundations of data exchange.** Cambridge University Press, 2014.

[BS81] François Bancilhon and Nicolas Spyratos. **"Update semantics of relational views".** *ACM Transactions on Database Systems (TODS)* 6.4 (1981), pp. 557–575.

[Cod70] Edgar F Codd. **"A relational model of data for large shared data banks".** *Communications of the ACM* 13.6 (1970), pp. 377–387.

[GK12] Nicola Gambino and Joachim Kock. **“Polynomial functors and polynomial monads”.** *Mathematical Proceedings of the Cambridge Philosophical Society* 154.1 (Sept. 2012), pp. 153–192. ISSN: 1469-8064. DOI: 10.1017/s0305004112000394. URL: http://dx.doi.org/10.1017/S0305004112000394.

[JD94] Michael Johnson and Christopher NG Dampney. **“On the value of commutative diagrams in information modelling”.** *Algebraic Methodology and Software Technology (AMAST’93) Proceedings of the Third International Conference on Algebraic Methodology and Software Technology, University of Twente, Enschede, The Netherlands 21–25 June 1993*. Springer. 1994, pp. 45–58.

[JR02] Michael Johnson and Robert Rosebrugh. **"Sketch data models, relational schema and data specifications".** *Electronic Notes in Theoretical Computer Science* 61 (2002), pp. 51–63.

[JRW00] Michael Johnson, Robert Rosebrugh, and RJ Wood. **"Entity-relationship models and sketches".** *Journal Theory and Applications of Categories* (2000).

[LS90] S Kazem Lellahi and Nicolas Spyratos. **"Towards a categorical data model supporting structured objects and inheritance".** *International East/West Database Workshop*. Springer. 1990, pp. 86–105.

[NS23] Nelson Niu and David I. Spivak. **Polynomial Functors: A Mathematical Theory of Interaction.** 2023. arXiv: 2312.00990 [math.CT].

[PLF22] Evan Patterson, Owen Lynch, and James Fairbanks. **"Categorical data structures for technical computing".** *Compositionality: the open-access journal for the mathematics of composition* 4 (2022).

[RMM24] Gabriel Goren Roig, Joshua Meyers, and Emilio Minichiello. **Presenting Profunctors.** 2024. arXiv: 2404.01406 [math.CT].

[Ros] Robert Rosebrugh. **Implementing database design (and manipulation) categorically.** URL: https://www.appliedcategorytheory.org/wp-content/uploads/2017/09/Rosebrugh-Implementing-database-design-and-manipulation-categorically.pdf.

[RW91] Robert Rosebrugh and RJ Wood. **"Relational databases and indexed categories".** *Proceedings of the International Category Theory Meeting 1991, CMS Conference Proceedings.* Vol. 13. 1991, pp. 391–407.

References v

[Sch+17] Patrick Schultz et al. **“Algebraic Databases”.** *Theory and Applications of Categories* 32.16 (2017), pp. 547–619.

[Sha21] Brandon Shapiro. **Familial Monads as Higher Category Theories.** 2021. arXiv: `2111.14796 [math.CT]`.

[Spi12] David I Spivak. **“Functorial data migration”.** *Information and Computation* 217 (2012), pp. 31–51.

[SW15] David I Spivak and Ryan Wisnesky. **“Relational foundations for functorial data migration”.** *Proceedings of the 15th Symposium on Database Programming Languages*. 2015, pp. 21–28.

[SW17] Patrick Schultz and Ryan Wisnesky. **“Algebraic data integration”.** *Journal of Functional Programming* 27 (2017).