Algebraic Databases

Patrick Schultz, David Spivak, Christina Vasilakopoulou
Department of Mathematics
Massachusetts Institute of Technology

Ryan Wisnesky
Categorical Informatics

NEPLS
October 7, 2016

$\mathcal{C}_i$

Outline

- In the early 2010s, Spivak proposed using the *functorial data model* (FDM) to solve data migration problems.
  - Schemas are categories, instances are set-valued functors.
  - A precursor to the FDM was proposed by Rosebrugh in the early 2000s.
- It turns out the FDM can be understood as algebraic specification.
- Talk goal: describe the FDM using algebraic terminology.
- The FDM is being commercialized in a data integration tool by an MIT spin-out company. Both the lab and the company are looking for collaborators. (catinf.com)
- Sponsored by ONR grant N000141310260, AFOSR grant FA9550-14-1-0031, NIST SBIR grants 70NANB15H290 and 70NANB16H178, and NSF I-Corps grant 1611699.

Review of Algebraic Theories (Signatures, Terms)

A *signature* Sig consists of a set of *sorts* and a set of *function symbols*

$$f\colon s_1 \times \ldots \times s_n \to s$$

A *context* Γ over signature Sig assigns variables to sorts

$$v_1\colon s_1,\ \ldots,\ v_k\colon s_k$$

A *term* of sort s in context Γ is either

- a variable v, if $v : s \in \Gamma$
- a function application $f(t_1, \ldots, t_n)$, if $f\colon s_1 \times \ldots \times s_n \to s$ and each t_i is a term of sort s_i

We write $t[v \mapsto t']$ for the substitution of term t' for variable v in term t.

Review of Algebraic Theories (Theories, Entailment)

Let Sig be a signature. A (universally quantified) *equation* over Sig is a formula $\Gamma.\ t = t' : s$, where t, t' have sort s in context Γ. A set of equations Th is a *theory*. The entailment relation $Th \vdash$ between equations is defined by inference rules

$$\frac{}{\Gamma.\ t = t : s} \qquad \frac{\Gamma.\ t = t' : s}{\Gamma.\ t' = t : s} \qquad \frac{\Gamma.\ t = t' : s \qquad \Gamma.\ t' = t'' : s}{\Gamma.\ t = t'' : s}$$

$$\frac{\Gamma.\ t = t' : s \qquad v \notin \Gamma}{\Gamma, v\colon s'.\ t = t' : s} \qquad \frac{\Gamma, v\colon s.\ t = t' : s' \qquad \Gamma.\ e = e' : s}{\Gamma.\ t[v \mapsto e] = t'[v \mapsto e'] : s'}$$

Review of Algebraic Theories (Theory Morphisms)

- A *morphism of signatures* $F : Sig_1 \to Sig_2$ consists of
  - a function from sorts in Sig_1 to sorts in Sig_2
  - a function from symbols $f\colon s_1 \times \ldots \times s_n \to s$ in Sig_1 to (open) terms $F(s_1) \times \ldots \times F(s_n) \to F(s)$ in Sig_2

- A *morphism of theories* $F : Th_1 \to Th_2$ is a morphism of signatures that preserves provable equality of terms

$$Th_1 \vdash v_1\colon s_1,\ \ldots,\ v_n\colon s_n.\ t_1 = t_2 : s$$

implies

$$Th_2 \vdash v_1\colon F(s_1),\ \ldots,\ v_n\colon F(s_n).\ F(t_1) = F(t_2) : F(s)$$

Review of Algebraic Theories (Models)

- An *algebra* A over signature Sig consists of
  - a set $A(s)$ for each sort s
  - a function $A(f) : A(s_1) \times \ldots \times A(s_k) \to A(s)$ for each symbol $f : s_1 \times \ldots \times s_k \to s$
- An *environment* η for context Γ takes each $v : s \in \Gamma$ to some $A(s)$.
- We write $A[\![t]\!]\eta$ for the meaning of term t in environment η.
- A is a *model* of a theory Th ($Th \models A$) when $Th \vdash \Gamma.\ t = t' : s$ implies $A[\![t]\!]\eta = A[\![t']\!]\eta$ for all terms t, t' and environments η.
- A *morphism* of Sig-algebras $h : A \to B$ is a family of functions $h(s) : A(s) \to B(s)$ such that

$$h(s)(A(f)(a_1, \ldots, a_n)) = B(f)(h(s_1)(a_1), \ldots, h(s_n)(a_n))$$

for every symbol $f : s_1 \times \ldots \times s_n \to s$ and $a_i \in A(s_i)$.

Review of Algebraic Theories (Key Properties)

- Entailment is semi-decidable.
- Deduction is sound and complete.
- Every theory Th admits a *term model* M:
  - $M(s)$ is the set of ground terms of sort s, modulo $Th \vdash$.
  - M is *initial*: for every $M' \models Th$, there is a unique $M \to M'$.
  - Construction of M is semi-computable.
- Algebraic theories are presentations of cartesian multi-categories.

Algebraic Databases (Typesides)

- A *typeside* Ty is an algebraic theory.
  - Its sorts are called *types*.
  - It represents an ambient computational context.

Example typeside

- Types

$$\mathsf{Nat},\ \mathsf{Char},\ \mathsf{String}$$

- Symbols

$$\mathsf{zero}\colon \mathsf{Nat},\ \mathsf{succ}\colon \mathsf{Nat} \to \mathsf{Nat},\ +\colon \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat}$$

$$\mathsf{A}, \mathsf{B}, \mathsf{C}, \ldots, \mathsf{Z} : \mathsf{Char}$$

$$\mathsf{nil}\colon \mathsf{String},\ \mathsf{cons}\colon \mathsf{Char} \times \mathsf{String} \to \mathsf{String}$$

- Equations

$$\forall x.\ +(\mathsf{zero}, x) = x$$

$$\forall x, y.\ +(\mathsf{succ}(x), y) = \mathsf{succ}(+(x, y))$$

Algebraic Databases (Schemas)

- A *typeside* Ty is an algebraic theory.
  - Its sorts are called *types*.
  - It represents an ambient computational context.
- A *schema* S on Ty extends Ty with
  - new sorts (called *entities*).
  - new symbols $att\colon entity \to type$ (called *attributes*).
  - new symbols $fk\colon entity \to entity$ (called *foreign keys*).
  - new unary equations of the form $\forall v\colon entity.\ e = e'$.

Example Schema

- Entities

$$\text{Emp},\ \text{Dept}$$

- Foreign Keys

$$\text{manager}\colon \text{Emp} \to \text{Emp},\ \text{works}\colon \text{Emp} \to \text{Dept},\ \text{secretary}\colon \text{Dept} \to \text{Emp}$$

- Attributes

$$\text{dname}\colon \text{Dept} \to \text{String},\ \text{ename}\colon \text{Emp} \to \text{String}$$

- Equations

$$\forall e.\ \text{works}(\text{manager}(e)) = \text{works}(e)$$

$$\forall d.\ \text{works}(\text{secretary}(d)) = d$$

$$\forall e.\ \text{manager}(\text{manager}(e)) = \text{manager}(e)$$

Algebraic Databases (Instances)

- A *typeside* Ty is an algebraic theory.
  - Its sorts are called *types*.
  - It represents an ambient computational context.

- A *schema* S on Ty extends Ty with
  - new sorts (called *entities*).
  - new symbols $att\colon entity \to type$ (called *attributes*).
  - new symbols $fk\colon entity \to entity$ (called *foreign keys*).
  - new unary equations of the form $\forall v\colon entity.\ e = e'$.

- An *instance* I on S extends S with
  - new symbols $gen\colon entity$ (called *generators*).
  - new symbols $sk\colon type$ (called *labelled nulls / skolem variables*).
  - new non-quantified equations.

Example Instance

- Generators

$$a, b, c\colon \mathsf{Emp},\ \ m, s\colon \mathsf{Dept}$$

- Equations

$$\mathsf{ename}(a) = \mathsf{Al},\ \ \mathsf{ename}(c) = \mathsf{Carl},\ \ \mathsf{dname}(m) = \mathsf{Math}$$

$$\mathsf{works}(a) = m,\ \ \mathsf{works}(b) = m,\ \ \mathsf{secretary}(s) = c,\ \ \mathsf{secretary}(m) = b$$

Abbreviated $\mathsf{cons}(\mathsf{A}, \mathsf{cons}(\mathsf{l}, \mathsf{nil}))$ as Al, etc.

Instance Semantics is its Initial Term Model

Dept		
ID	**dname**	**secretary**
m	Math	b
s	dname(s)	c

Emp			
ID	**ename**	**manager**	**works**
a	Al	mgr(a)	m
b	ename(b)	mgr(b)	m
c	Carl	mgr(c)	s
mgr(a)	ename(mgr(a))	mgr(a)	m
mgr(b)	ename(mgr(b))	mgr(b)	m
mgr(c)	ename(mgr(c))	mgr(c)	s

Functorial Data Migration

- Let S, T be two schemas on typeside Ty. A *morphism* $F\colon S \to T$ is a morphism of theories **that is the identity on** Ty.
  - The schemas on Ty form a category.

- Let I, J be two instances on schema S. A *morphism* $h\colon I \to J$ is a morphism of theories **that is the identity on** S.
  - The instances on S form a category, S-inst.

- A morphism $F\colon S \to T$ induces adjoint *data migration functors*
  - $\Sigma_F\colon S$-inst$\to T$-inst (like outer disjoint union then quotient)

    defined as substitution

  - $\Delta_F\colon T$-inst$\to S$-inst (like project)

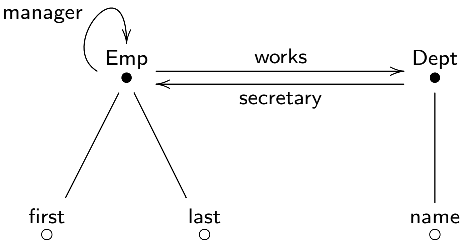
    $$\Sigma_F \dashv \Delta_F$$

  - $\Pi_F\colon S$-inst$\to T$-inst (like join)

    $$\Delta_F \dashv \Pi_F$$

Note: adjoints are only defined up to unique isomorphism.

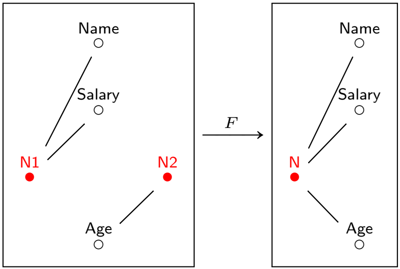
Graphical (E/R Diagram) Notation for Schemas



$$\mathsf{Emp.manager.works} = \mathsf{Emp.works} \qquad \mathsf{Dept.secretary.works} = \mathsf{Dept}$$

$$\mathsf{Emp.manager.manager} = \mathsf{Emp.manager}$$

Δ (Project)



N1		
ID	**Name**	**Salary**
1	Alice	$100
2	Bob	$250
3	Sue	$300

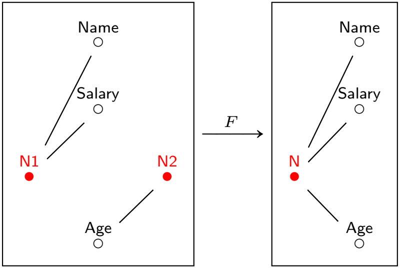
N2	
ID	**Age**
4	20
5	20
6	30

$\xleftarrow{\Delta_F}$

N			
ID	**Name**	**Salary**	**Age**
a	Alice	$100	20
b	Bob	$250	20
c	Sue	$300	30

In these examples we show instances as term models rather than theories. IDs are meaningless – instances are only defined up to isomorphism.

Π (Join)



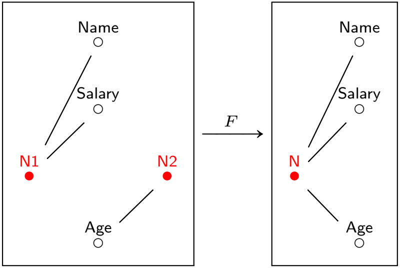
N1		
ID	**Name**	**Salary**
1	Alice	\$100
2	Bob	\$250
3	Sue	\$300

N2	
ID	**Age**
4	20
5	20
6	30

$\xrightarrow{\Pi_F}$

N			
ID	**Name**	**Salary**	**Age**
a	Alice	\$100	20
b	Alice	\$100	20
c	Alice	\$100	30
d	Bob	\$250	20
e	Bob	\$250	20
f	Bob	\$250	30
g	Sue	\$300	20
h	Sue	\$300	20
i	Sue	\$300	30

Σ (Outer Disjoint Union then Quotient)



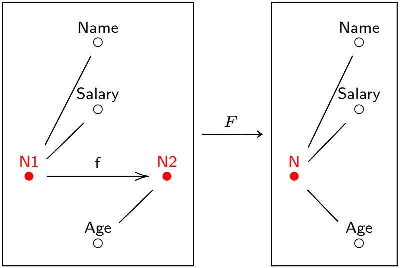
N1			N2	
ID	**Name**	**Salary**	**ID**	**Age**
1	Alice	\$100	4	20
2	Bob	\$250	5	20
3	Sue	\$300	6	30

$\xrightarrow{\Sigma_F}$

N			
ID	**Name**	**Salary**	**Age**
a	Alice	\$100	$null_1$
b	Bob	\$250	$null_2$
c	Sue	\$300	$null_3$
d	$null_4$	$null_5$	20
e	$null_6$	$null_7$	20
f	$null_8$	$null_9$	30

($null_1$ abbreviates Age(a), etc.)

Foreign keys



N1			
ID	**Name**	**Salary**	**f**
1	Alice	$100	4
2	Bob	$250	5
3	Sue	$300	6

N2	
ID	**Age**
4	20
5	20
6	30

$\xleftarrow{\Delta_F}$

$\xrightarrow{\Pi_F, \Sigma_F}$

N			
ID	**Name**	**Salary**	**Age**
a	Alice	$100	20
b	Bob	$250	20
c	Sue	$300	30

Expressivity of Δ, Σ, Π

- Data migrations of the form

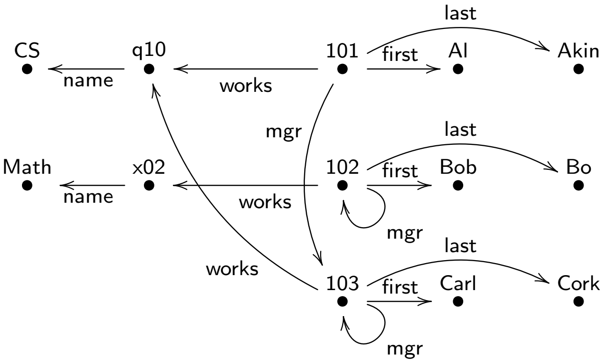
$$\Sigma_F \circ \Delta_G \circ \Pi_H$$

  can express any SPCU relational algebra query under **bag** semantics. SPCU under **set** semantics can express the data migration when
  - F is a discrete op-fibration (ensures union compatibility).
  - H is a surjection on attributes (ensures domain independence).
  - All theories denote finite categories (ensures computability).
  - The typeside has no function symbols (ensures atomicity of data).
  - We extend SPCU with a key generator (need fresh constants).

- Σ_F has similar semantics to an operation called the *chase* which is the basis of relational data integration.

- Migrations of the form $\Delta_F \circ \Pi_G$ and $\Sigma_G \circ \Delta_F$ can be specified using for/where/return syntax.

Pivot (Instance ⇔ Schema)



Emp				
ID	**mgr**	**works**	**first**	**last**
101	103	q10	Al	Akin
102	102	x02	Bob	Bo
103	103	q10	Carl	Cork

Dept	
ID	**name**
q10	CS
x02	Math

QINL vs LINQ

- Functorial data migration is a *QINL* (i.e., co-LINQ) query mechanism.
- LINQ enriches programs with (schemas, queries and instances).
  - Collections are *terms*

$$\text{Employee} \colon \text{Set Int} \quad \text{manager} \colon \text{Set } (\text{Int} \times \text{Int})$$

  - e: Employee *is not* a judgment.
  - There *is* a term $\in \colon \text{Int} \times \text{Set Int} \to \text{Bool}$.
- QINL enriches (schemas, queries and instances) with programs.
  - Collections are *types*

$$\text{Employee} \colon \text{Type} \quad \text{manager} \colon \text{Employee} \to \text{Employee}$$

  - e: Employee *is* a judgment.
  - There *is not* a term $\in \colon \text{Employee} \times \text{Type} \to \text{Bool}$.
- LINQ is more popular, but QINL is common in Coq, Agda, etc.

QINL is “one level up” from LINQ

- LINQ
  - Schemas are collection types over a base type theory
    $$\mathsf{Set}\ (\mathsf{Int} \times \mathsf{String})$$
  - Instances are terms
    $$\{(1, \mathsf{CS})\} \cup \{(2, \mathsf{Math})\}$$
  - Data migrations are functions
    $$\pi_1 \colon \mathsf{Set}\ (\mathsf{Int} \times \mathsf{String}) \to \mathsf{Set}\ \mathsf{Int}$$
- QINL
  - Schemas are type theories over a base type theory
    $$\mathsf{Dept},\ \mathsf{name} \colon \mathsf{Dept} \to \mathsf{String}$$
  - Instances are term models
    $$\mathsf{d}_1, \mathsf{d}_2 \colon \mathsf{Dept},\ \mathsf{name}(\mathsf{d}_1) = \mathsf{CS},\ \mathsf{name}(\mathsf{d}_2) = \mathsf{Math}$$
  - Data migrations are functors
    $$\Delta_{\mathsf{Dept}} \colon (\mathsf{Dept}, \mathsf{name} \colon \mathsf{Dept} \to \mathsf{String})\text{-}\mathsf{inst} \ \to\ (\mathsf{Dept})\text{-}\mathsf{inst}$$

FQL Demo

- The FQL IDE is an open-source graphical schema mapping and data integration tool available at

  categoricaldata.net/fql.html

- It is being commercialized by Categorical Informatics, a recent MIT spin out

  catinf.com

Conclusion

- I presented an expressive formalism for specifying and manipulating databases using algebraic theories.
- Many concepts from algebraic specification appear in this work. Some not mentioned:
  - Conservative extensions / consistency
  - Institutions, limits, colimits, etc
  - Automated theorem proving
- Looking for feedback, users, and collaborators.

catinf.com