Computational Schemas

Categorical Informatics, Inc.

info@catinf.com

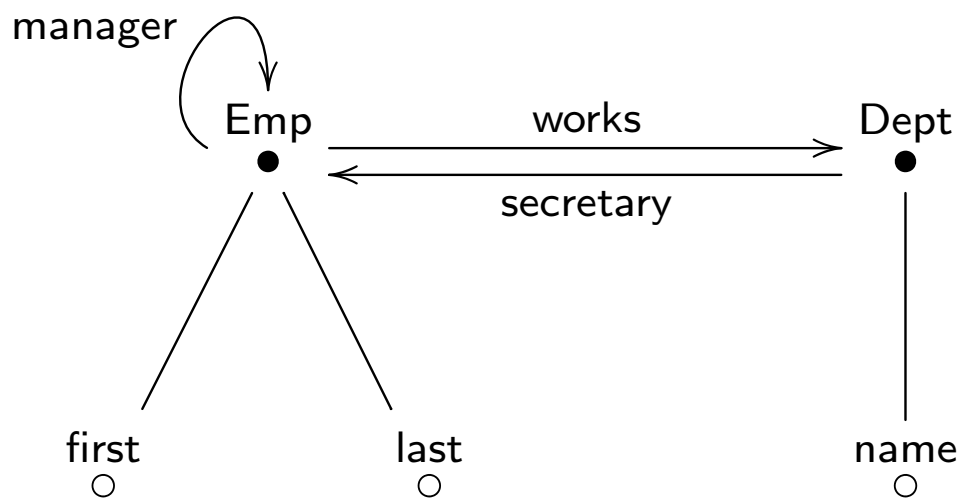
May 1, 2016

$$\mathcal{C}_i$$

Outline

- Review of the Functorial Data Model
  - Schemas are categories, instances are functors.
- This talk:
  - More recent languages in the FQL family can *compute* in schemas, queries, migrations, etc.
  - Computational primitives are user-defined using equational logic, and, optionally, a java binding.
  - Examples: boolean algebra, arithmetic, strings, images, machine learning, entity resolution.
  - Our approach is a kind of dual to LINQ; see `http://arxiv.org/abs/1511.06459`.
  - See `http://arxiv.org/abs/1503.03571` for more examples.

Review of the Functorial Data Model



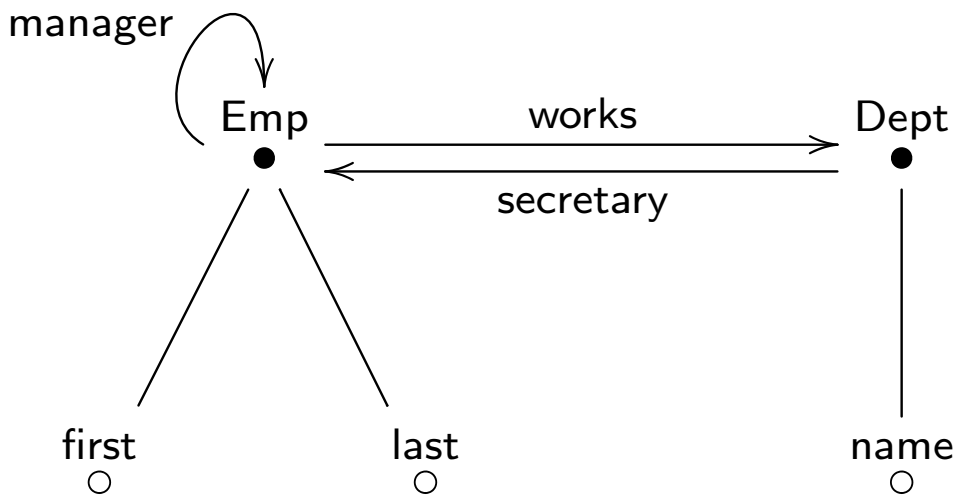
Emp.manager.works = Emp.works Dept.secretary.works = Dept

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

Dept		
ID	**sec**	**name**
q10	102	CS
x02	101	Math

Schemas as Equational Theories

- We call the graph of a schema its *signature*.



- In this talk we will write the path equivalences in a schema:

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

using logical notation:

$$\forall e : \mathsf{Emp}.\ \mathsf{works}(\mathsf{manager}(e)) = \mathsf{works}(e) \qquad \forall d : \mathsf{Dept}.\ \mathsf{works}(\mathsf{secretary}(d)) = d$$

- Attributes and foreign keys are unary function symbols:

$$\mathsf{name} : \mathsf{Dept} \to \mathsf{String} \qquad \mathsf{manager} : \mathsf{Emp} \to \mathsf{Emp} \qquad etc.$$

The Type Side

- A schema exists in an ambient equational theory, called the *type side*.
- For example, a type side for arithmetic has function symbols:

$$0 : \mathsf{Nat} \qquad S : \mathsf{Nat} \to \mathsf{Nat} \qquad + : \mathsf{Nat}, \mathsf{Nat} \to \mathsf{Nat}$$

  and equations:

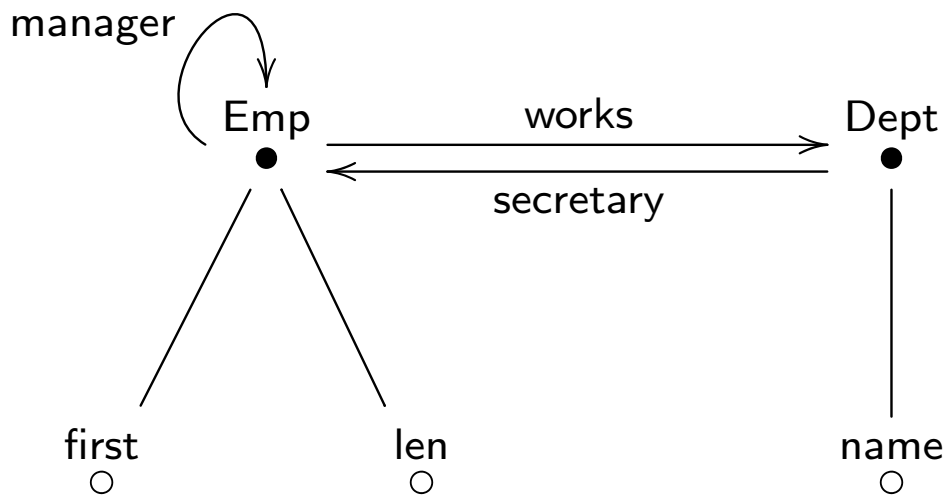
$$\forall n : \mathsf{Nat}.\ n + 0 = n \qquad \forall n, m : \mathsf{Nat}.\ n + (S(m)) = S(n + m)$$

- We will assume function symbols for string manipulation, including:

$$\mathsf{length} : \mathsf{String} \to \mathsf{Nat}$$

Computations in Schemas

- We can use a type side when we define a schema:



$\forall e$: Emp. works(manager(e)) = works(e) $\forall d$: Dept. works(secretary(d)) = d

$\forall e$: Emp. len(e) = length(first(e))

Emp				
ID	**mgr**	**works**	**first**	**len**
101	103	q10	Al	2
102	102	x02	Bob	3
103	103	q10	Carl	4

Dept		
ID	**sec**	**name**
q10	102	CS
x02	101	Math

Derived Columns

- Every "partial" instance on a schema has a unique "completion".
- So an equation such as

$$\forall e : \mathsf{Emp}.\ \mathsf{len}(e) = \mathsf{length}(\mathsf{first}(e))$$

  will cause the FQL tool to compute the len attribute from the first attribute whenever the len attribute value is missing in an instance.
- The FQL tool never requires complete instances, since it can always complete a partial instance.

Example Partial Instance

- Partial Instance (Dept table omitted):

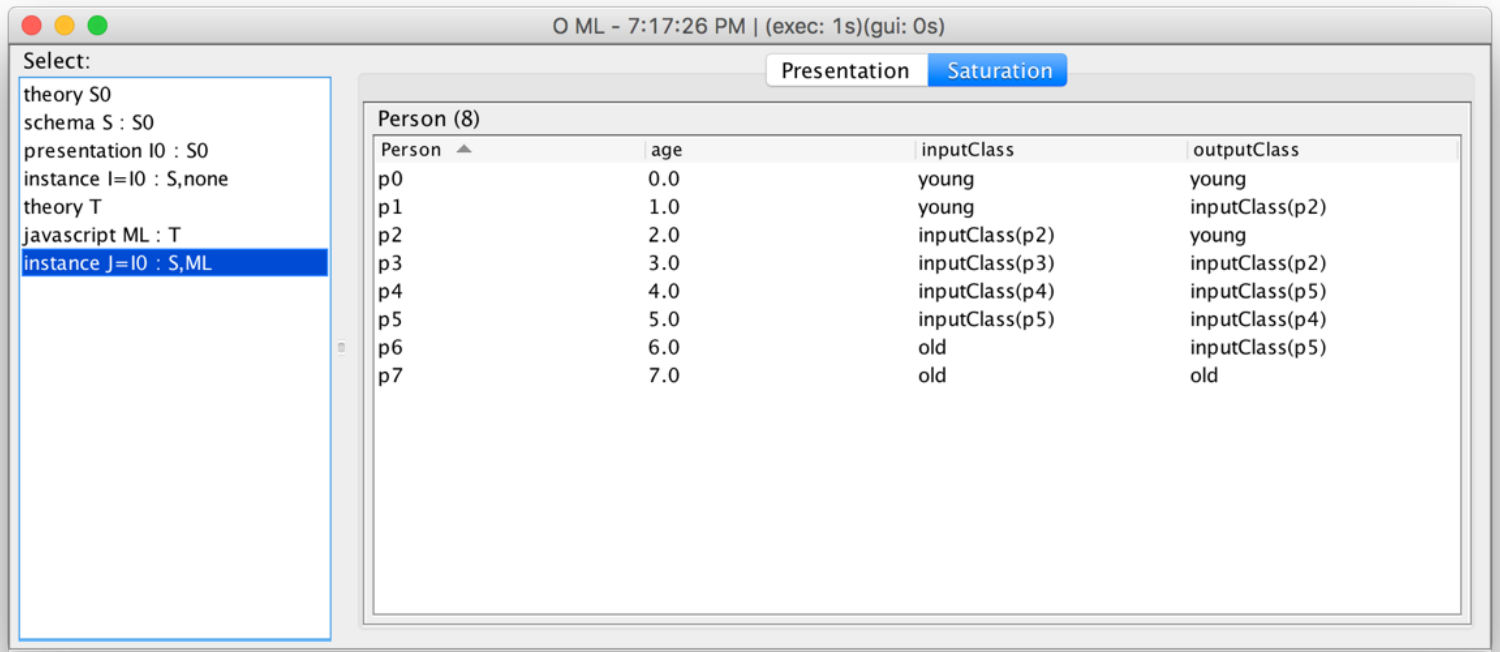
Emp				
ID	**mgr**	**works**	**first**	**last**
101	103	q10		Akin
102	102	x02	Bob	Bo
103		q10	Carl	Cork

- Completion (Dept table omitted):

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

Binding Primitives to Existing Code

- Function symbols can be bound to user-defined java functions.
- Here is a screen shot of an FQL instance that invokes a 3rd party machine learning library to classify input ages into old and young:



- This example illustrates the interplay between computation and unknown values.

Conclusion

- All schemas live in a user-defined computational context.
- Computational primitives, attributes, and foreign keys can mix freely and can optionally be bound to user-provided java code.
- Columns to be derived from other columns on the fly.
- Computation and unknown values work together.