Databases are Categories II

Refinements and Extensions

David I. Spivak

dspivak@math.mit.edu
Mathematics Department
Massachusetts Institute of Technology

Presented on 2010/10/22

Last time

- On June 3, 2010 I gave a talk here at Galois called “Databases are categories.”
- The main idea was that:
    - a database schema is a category $\mathcal{C}$, and
    - a state on that schema is a functor $\delta\colon \mathcal{C} \longrightarrow \mathbf{Set}$.
- Here is a quick review:

Review of basic category theory

- A category $\mathcal{C}$ is a system of objects and arrows, and a composition law.
- A functor $\mathcal{C} \to \mathcal{D}$ is a mapping that preserves these structures.
- **Set** is the category whose objects are sets, whose arrows are (total) functions, and whose composition law is the specification of how functions compose.
- If $\mathcal{C}$ is the category on the left below, then a functor $\delta \colon \mathcal{C} \to \textbf{Set}$ might look like this:

$$\mathcal{C} := \boxed{\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "g"'] & B \\ C & \end{tikzcd}} \;; \qquad \delta = \boxed{\begin{tikzcd} \boxed{\bullet a_1 \;\; \bullet a_2 \;\; \bullet a_3} \arrow[r, "{(b_1,b_1,b_2)}"] \arrow[d] & \boxed{\bullet b_1 \; \bullet b_2} \\ \boxed{\bullet c_1} & \end{tikzcd}}$$

How databases fit in

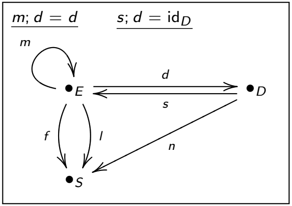
A category $\mathcal{C}$ as a schema: Each object $A \in \mathbf{Ob}(\mathcal{C})$ is a table, each arrow $A \to B$ is a foreign key column of table A pointing to table B.

Employee				
Id	**First**	**Last**	**Mgr**	**Dpt**
101	David	Hilbert	103	q10
102	Bertrand	Russell	102	x02
103	Alan	Turing	103	q10

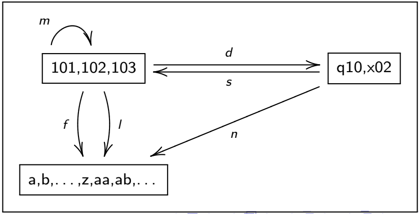
Department		
Id	**Name**	**Secr'y**
q10	Sales	101
x02	Production	102

String
Id
a
b
⋮
z
aa
ab
⋮

$\mathcal{C} =$



$\delta : \mathcal{C} \to \mathbf{Set}$



Purpose of this talk

The purpose of this talk is:

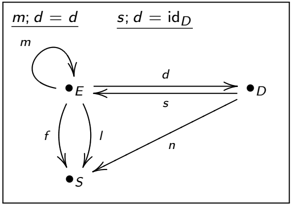
- to refine the above “databases are categories” notion a bit:
  - a database schema is really “a sketch.”
    In other words, a database schema is a presentation of a category.
  - We can use a similar idea to discuss incomplete data.
- to discuss querying and data migration.
- to discuss typing and calculated fields.
- to talk about ontologies, or “ologs,” and a vision of “functorial communication” based on the above ideas.

Subtlety in “database schema=category” idea

Question: So, what’s wrong with saying that the schema for the database on the left is the category on the right?

Employee				
Id	**First**	**Last**	**Mgr**	**Dpt**
101	David	Hilbert	103	q10
102	Bertrand	Russell	102	x02
103	Alan	Turing	103	q10

Department		
Id	**Name**	**Secr’y**
q10	Sales	101
x02	Production	102



$m; d = d$ $s; d = \mathrm{id}_D$

Answer:

- Categories have a composition law defined for every pair of composable arrows.
- But in the picture on the right, we don’t have an arrow for $s; f$.
- Similarly in the schema on the left, there is no “secretary’s first name” column in the department table.
- Conclusion: we are only imposing a composition law where we need to.

Why venture into sketches?

- We want to be able to write down a set of objects, arrows, and composition laws, without having to throw in an arrow for every possible composition. How do we encode that?
- We want to present categories by generators (objects and arrows) and relations (composition laws).
- We want to "sketch a category."

Information schemas

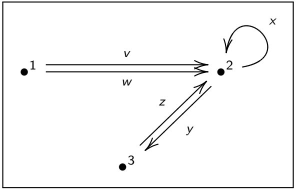
Suppose we don't have any composition laws to speak of. What information do we need in order to specify such a schema?
Information schema:

$$\mathcal{G} := \boxed{\begin{tikzcd} \bullet\text{Table} & \bullet\text{Column} \arrow[l, bend right, "\text{in}"'] \arrow[l, bend left, "\text{foreign\_table}"] \end{tikzcd}} = \boxed{\begin{tikzcd} \bullet\text{Object} & \bullet\text{Arrow} \arrow[l, bend right, "\text{source}"'] \arrow[l, bend left, "\text{target}"] \end{tikzcd}}$$

Here's an example state on this schema, a functor $\delta \colon \mathcal{G} \to \mathbf{Set}$:

Object
Id
1
2
3

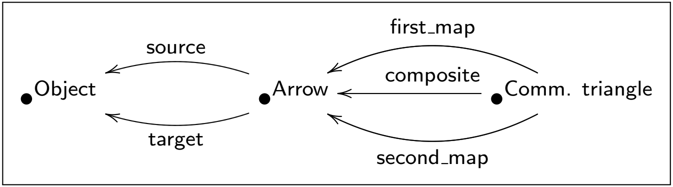
Arrow		
Id	**source**	**target**
v	1	2
w	1	2
x	2	2
y	2	3
z	3	2



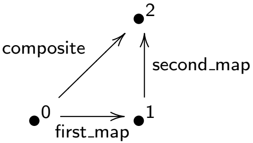
Information schema for category sketches

Schema for category sketches:

$\mathbf{\Delta} :=$



Here we would need three additional equations coming from the triangle



first_map;source = composite;source

first_map;target = second_map;source

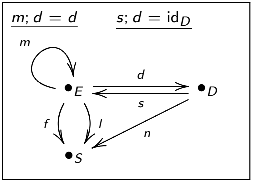
second_map;target = composite;target

Sketching our example, $\mathcal{C}$

- $\mathcal{C}$ is a schema for a department store.
- But how do we record the schema $\mathcal{C}$ itself, in terms of the information schema $\mathbf{\Delta}$?

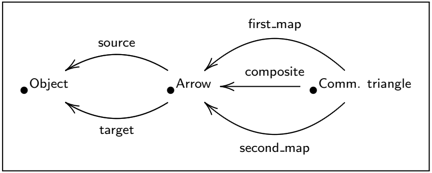
Record

$\mathcal{C} =$



in terms of

$\mathbf{\Delta} =$



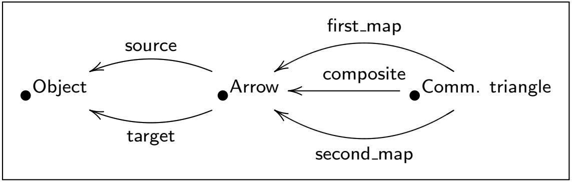
Object
Id
E
D
S

Arrow		
Id	**source**	**target**
m	E	E
d	E	D
s	D	E
f	E	S
l	E	S
n	D	S
i	D	D

Comm. triangle			
Id	**first_map**	**second_map**	**composite**
1	m	d	d
2	s	d	i
3	d	i	d
4	i	s	s
5	i	n	n
6	i	i	i

Conclusion

$\mathbf{\Delta} :=$



- We can write any schema $\mathcal{C}$ as a database state on $\mathbf{\Delta}$.
- We refine our definition from last time to say that a database schema is a state on $\mathbf{\Delta}$.
- I'll call these "pre-categories." They are basically equivalent to categories.
- A functor between pre-categories is just a morphism of states on $\mathbf{\Delta}$.
- I swept the difference under the rug last time because I wanted to emphasize the tight connection between database schemas and categories. That connection is still tight.

But once we're here...

- Now that we're working with presentations of categories rather than categories,
- it might be nice to "present" other facts, besides composition laws.
- We can use more complex information schemas (beef up $\mathbf{\Delta}$) to specify that a certain table in our schema $\mathcal{C}$
  - is a product of other tables,
  - is a fiber product,
  - is a colimit,
  - is an exponential object,
  - is empty (i.e. has no rows),
  - has only one row,
  - etc.
- We can do all this with sketches.

Sketches and sketch maps

- "Sketch" is category-theory terminology for "category specification."
- In a sketch we can specify that a certain object must be the limit or the colimit of some diagram.
- This could be used, e.g., in a database where we want to have a table of "airplane seats" which is the coproduct of the tables "first class seats" and "economy class seats".
- We specify that we want $S = F \amalg E$ in the sketch $\mathcal{C}$.
- Now, instead of states being functors $\mathcal{C} \to \textbf{Set}$, states are "sketch maps" $\delta \colon \mathcal{C} \to \textbf{Set}$; i.e. functors that preserve all the specified facts.
- For example for $\delta \colon \mathcal{C} \to \textbf{Set}$ to be a sketch map, we must have

$$\delta(S) = \delta(F) \amalg \delta(E).$$

Language of sketches as formal UML

- At this point, one recognizes that sketches are quite similar to UML (Unified Modeling Language) diagrams for database schemas.
- You just specify what you want.
- What's new here is the connection between database theory and category theory.
- Category theory brings formal reasoning to the picture that was already there.

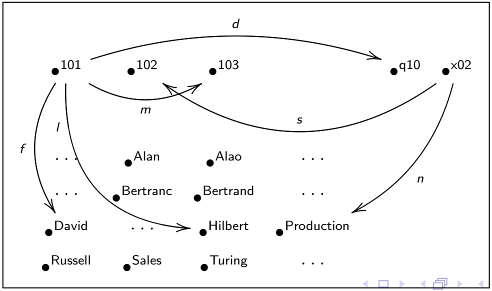
Sketching states with “RDF triple stores”

- Recall from the first talk that given a category $\mathcal{C}$ and a functor $\delta\colon \mathcal{C} \longrightarrow \textbf{Set}$ one can take the Grothendieck construction $Gr(\delta)$.
- Suppose given the following example, considered as $\delta\colon \mathcal{C} \longrightarrow \textbf{Set}$

Employee				
Emp_Id	**First**	**Last**	**Mgr**	**Dpt**
101	David	Hilbert	103	q10
102	Bertrand	Russell	102	x02
103	Alan	Turing	103	q10

Department		
Dept_Id	**Name**	**Secr'y**
q10	Sales	101
x02	Production	102

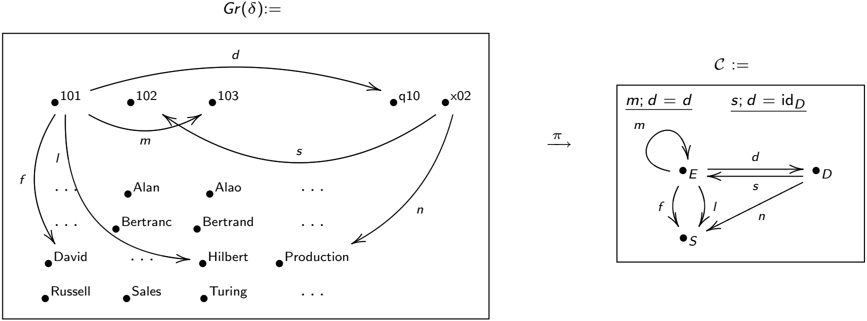
Applying the Grothendieck construction, we get a category $Gr(\delta)$:



Change of perspective

Given $\delta\colon \mathcal{C} \to \mathbf{Set}$, the Grothendieck construction of δ gives a functor

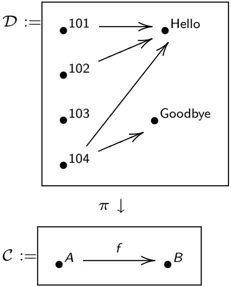
$$\pi\colon Gr(\delta) \to \mathcal{C}.$$



The fiber over (inverse image of) every object $X \in \mathcal{C}$ is a set of objects in $\pi^{-1}(X) \in Gr(\delta)$. That set is $\delta(X)$.

Allowing for incomplete, non-atomic, or bad data

- We can think of any functor $\pi\colon \mathcal{D} \to \mathcal{C}$ as a "pre-state" on $\mathcal{C}$.
- Such a functor π can encode incomplete, non-atomic, or bad data.



- Row 103 has no data in the f cell, and row 104 has too much.
- Bad data (data not conforming to declared composition laws) can also be encoded as a functor $\pi\colon \mathcal{D} \to \mathcal{C}$.
- Any pre-state on $\mathcal{C}$ can be "corrected" in a canonical way to a state.
- Conclusion: we can use category theory as a model even when things are awry.

Data migration

- Let's go back to the simple picture of database schemas as categories or pre-categories $\mathcal{C}$ and states as functors $\delta \colon \mathcal{C} \to \mathbf{Set}$. (Not sketchy).
- Given a schema $\mathcal{C}$, the category of states on $\mathcal{C}$ is denoted $\mathcal{C}$–$\mathbf{Set}$.
- Given a morphism between schemas, we want to be able to move data back and forth in canonical ways.
- That means, given $F \colon \mathcal{C} \to \mathcal{D}$ we want functors

$$\mathcal{C}\text{–}\mathbf{Set} \to \mathcal{D}\text{–}\mathbf{Set}$$

and

$$\mathcal{D}\text{–}\mathbf{Set} \to \mathcal{C}\text{–}\mathbf{Set}.$$

The "easy" migration functor, Δ

- Given a morphism of schemas (i.e. a functor)

$$F\colon \mathcal{C} \to \mathcal{D},$$

  we can transform states on $\mathcal{D}$ to states on $\mathcal{C}$ as follows:

$$(\delta\colon \mathcal{D} \to \mathbf{Set}) \mapsto ((\delta \circ F)\colon \mathcal{C} \to \mathbf{Set})$$

- Thus we have a functor $\Delta_F\colon \mathcal{D}\text{–}\mathbf{Set} \to \mathcal{C}\text{–}\mathbf{Set}$.
- Δ_F basically operates by "re-indexing." Using it, one can duplicate or drop tables or columns.

The "harder" migration functors, Σ and Π

Given a morphism of schemas (i.e. a functor) $F\colon \mathcal{C} \to \mathcal{D}$,

- the functor $\Delta_F\colon \mathcal{D}\text{–}\mathbf{Set} \to \mathcal{C}\text{–}\mathbf{Set}$ has two adjoints:
  - a left adjoint $\Sigma_F\colon \mathcal{C}\text{–}\mathbf{Set} \to \mathcal{D}\text{–}\mathbf{Set}$, and
  - a right adjoint $\Pi_F\colon \mathcal{C}\text{–}\mathbf{Set} \to \mathcal{D}\text{–}\mathbf{Set}$,

$$\begin{tikzcd}
\mathcal{C}\text{–}\mathbf{Set} \arrow[r, shift left, "\Sigma_F"] & \mathcal{D}\text{–}\mathbf{Set} \arrow[l, shift left, "\Delta_F"] \arrow[r, shift left, "\Delta_F"] & \mathcal{C}\text{–}\mathbf{Set} \arrow[l, shift left, "\Pi_F"]
\end{tikzcd}$$

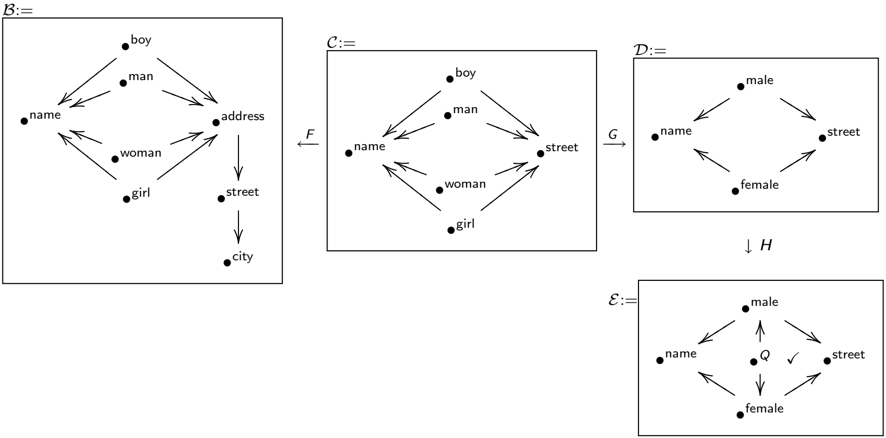
- Thus, given a morphism F of schemas, these three functors,

$$\Delta_F, \Sigma_F, \text{ and } \Pi_F$$

allow one to move data back and forth between $\mathcal{C}$ and $\mathcal{D}$ in canonical ways.

Views as polynomial functors

These functors can be arbitrarily composed to create views.



Given a state $\gamma\colon \mathcal{B} \to \mathbf{Set}$, what is $\Pi_H \circ \Sigma_G \circ \Delta_F(\gamma)\colon \mathcal{E} \to \mathbf{Set}$?

A simple "SELECT" query

SELECT title, isbn
FROM book
WHERE price $>$ 100

$\mathcal{C} :=$

$$\begin{tikzcd}
 & \bullet\text{book} \arrow[r, "\text{title}"] \arrow[d, "\text{price}"] \arrow[dr, "\text{isbn}"] & \bullet\text{String} \\
\bullet\mathbb{R}_{>100} \arrow[r] & \bullet\mathbb{R} & \bullet\text{isbn-num}
\end{tikzcd}$$

$\xrightarrow{F}$

$\mathcal{D} :=$

$$\begin{tikzcd}
\bullet W \arrow[r] \arrow[d] \arrow[dr, phantom, "\checkmark"] & \bullet\text{book} \arrow[r, "\text{title}"] \arrow[d, "\text{price}"] \arrow[dr, "\text{isbn}"] & \bullet\text{String} \\
\bullet\mathbb{R}_{>100} \arrow[r] & \bullet\mathbb{R} & \bullet\text{isbn-num}
\end{tikzcd}$$

$\xleftarrow{G}$

$\mathcal{E} :=$

$$\begin{tikzcd}
\bullet W \arrow[r, "\text{title}"] \arrow[dr, "\text{isbn}"] & \bullet\text{String} \\
 & \bullet\text{isbn-num}
\end{tikzcd}$$

$\Delta_G \circ \Pi_F$ is the appropriate view.
For any $\delta\colon \mathcal{C} \to \textbf{Set}$, we materialize the view as $\Delta_G \circ \Pi_F(\delta)$.

Interfacing between schemas

- We are often interested in taking data from one enterprise model $\mathcal{C}$ and transferring it to another enterprise model $\mathcal{D}$.
- Such transfers can also be accomplished using polynomial functors.
- However, if $\mathcal{C}$ and $\mathcal{D}$ are not basic schemas (they're too "sketchy") then the "harder" migration functors, Σ and Π, might not exist.
- Also, we might need to perform calculations such as concatenation, addition, comparison, conversion of units, etc. in order to interface these schemas.
- For this we'll need an underlying typing category.

Incorporating data types and functions

- In the example:

$$\mathcal{C} := \boxed{\begin{tikzcd} & \bullet^{\text{book}} \arrow[r, "\text{title}"] \arrow[d, "\text{price}"] \arrow[dr, "\text{isbn}"] & \bullet^{\text{String}} \\ \bullet^{\mathbb{R}_{>100}} \arrow[r] & \bullet^{\mathbb{R}} & \bullet^{\text{isbn-num}} \end{tikzcd}}$$

  how do we know that $\bullet^{\text{String}}$, $\bullet^{\mathbb{R}}$, and $\bullet^{\mathbb{R}_{>100}}$ are what they say they are?

- That is, given $\delta\colon \mathcal{C} \to \textbf{Set}$, how do we specify that $\delta(\bullet^{\mathbb{R}}) \in \textbf{Set}$ is some pre-defined data type like Float.

Power of category theory: connection is easy

- Let **Hask** denote a category of types and functions that has been implemented on a computer and for which (at least theoretically) there exists a functor $V\colon \mathbf{Hask} \to \mathbf{Set}$.
    - Think of **Hask** as all Haskell data types and the definable functions between them, as well as all new types that could possibly be output by modules.
- Now **Hask** begins to look like a schema and V a "canonical state."
- Since database schemas are categories and **Hask** is a category, we can integrate the two.

Example

- Here a nice functor

$\mathcal{B} :=$

$$\begin{tikzcd} & & \bullet^{\text{String}} \\ \bullet^{\mathbb{R}_{>100}} \arrow[r] & \bullet^{\mathbb{R}} & \end{tikzcd}$$

$$\xrightarrow{F} \textbf{Hask}$$

$$F(\bullet^{\text{String}}) = \text{char}(40), \quad F(\bullet^{\mathbb{R}}) = \text{Float}, \quad F(\bullet^{\mathbb{R}_{>100}}) = \text{some new type}.$$

- There is also an obvious functor

$\mathcal{B} =$

$$\begin{tikzcd} & & \bullet^{\text{String}} \\ \bullet^{\mathbb{R}_{>100}} \arrow[r] & \bullet^{\mathbb{R}} & \end{tikzcd}$$

$$\xrightarrow{G}$$

$$\begin{tikzcd} & \bullet^{\text{book}} \arrow[r, "\text{title}"] \arrow[d, "\text{price}"] \arrow[dr, "\text{isbn}"] & \bullet^{\text{String}} \\ \bullet^{\mathbb{R}_{>100}} \arrow[r] & \bullet^{\mathbb{R}} & \bullet^{\text{isbn-num}} \end{tikzcd}$$

$= \mathcal{C}$

- We are interested in functors $\delta \colon \mathcal{C} \to \textbf{Set}$ equipped with a map $\Delta_G \delta \to \Delta_F V$.

Typing in general

- If we need to enforce data types, our schema $\mathcal{C}$ will more than just a category (or sketch),
- It will be a category (or sketch) plus something like above:

$$\mathbf{Hask} \xleftarrow{F} \mathcal{B} \xrightarrow{G} \mathcal{C}.$$

- And we won't interested in any old state $\delta \colon \mathcal{C} \to \mathbf{Set}$ but only those with a map $\Delta_G\delta \to \Delta_F V$, where $V \colon \mathbf{Hask} \to \mathbf{Set}$ is as above.
- By definition of adjunction, that's just

$$\delta \to \Pi_G \Delta_F(V),$$

  and $\Pi_G\Delta_F(V)$ is some huge fixed state on $\mathcal{C}$ that encodes our typing requirements.
- So the category of typed states is $\mathcal{C}$–$\mathbf{Set}_{/\Pi_G\Delta_F(V)}$. This is a topos.

Summary

- The longer portion of this talk is over.
- I discussed how to tighten the connection between databases and mathematics.
    - How to only include the columns and composition laws you want (not necessarily all of them).
    - How to force tables to be products or coproducts (etc.) of other tables.
    - How to model incomplete, non-atomic, or bad data.
    - How functors between schemas give rise to views and data migration.
    - How to encode typing information and calculated fields to database schemas.

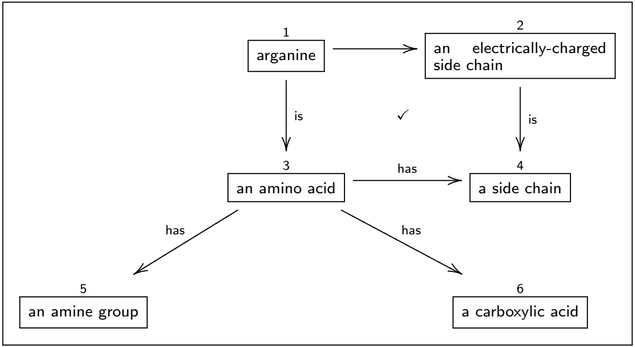
Bringing these concepts to researchers

- In this short second section, I want to look at these categories and sketches from a different perspective.
- Issue: how to allow people (e.g. scientists) to record very precise conceptual ideas.
    - A big problem in science is to know where data comes from and how its been manipulated to arrive at conclusions.
    - Another is to have the flexibility to design new databases on the fly, so as to capture unexpected nuances of data.
    - Finally, academic researchers need to be able to record precisely what they mean in a computer-searchable way, rather than in silos of prose.
- We need to democratize information storage.

Ologs: linguistic categories

- The word "Olog"
  - From "ontology log," a play on "Blog."
  - Alternatively, a suffix for "study of."
- An olog is a category or sketch, but with natural language labels.
- The goal is to capture meaning using explicit structure.

Example olog



1 arganine

2 an electrically-charged side chain

is

✓

is

3 an amino acid

has

4 a side chain

has

has

5 an amine group

6 a carboxylic acid

"Arganine is an amino acid. An amino acid has a side chain, a carboxylic acid, and an amine group. An electrically-charged side chain is a side chain. Arganine's side chain is electrically-charged."

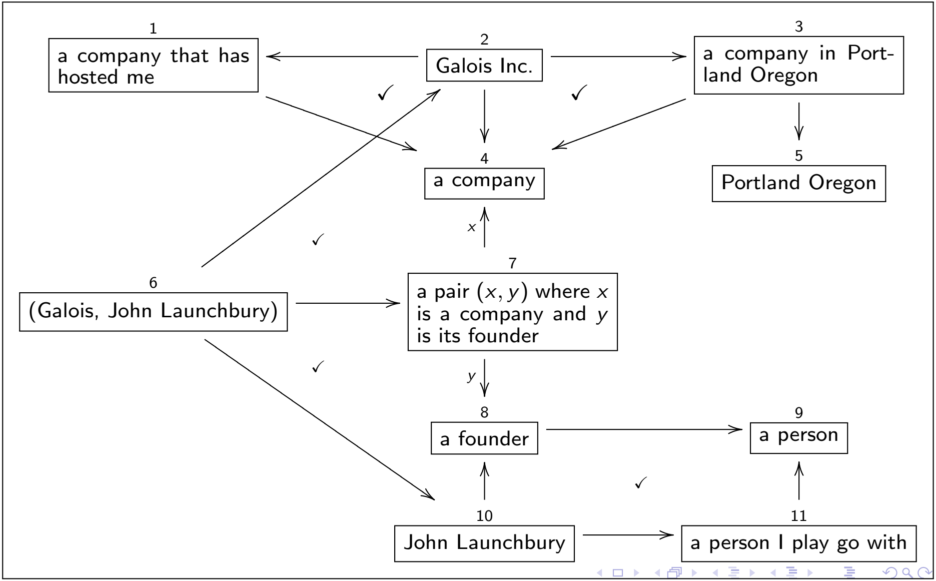
Another example

(1,2,4,5) and (2,3,5,6) are fiber products.

$$\begin{tikzcd}
\overset{1}{\boxed{\text{a material } x \text{ at room temperature}}} \arrow[r] \arrow[d, "\text{is}"] & \overset{2}{\boxed{\text{room temperature}}} \arrow[r, "\text{in kelvin}"] \arrow[d, "\text{is}"] & \overset{3}{\boxed{\{x \in \mathbb{R} : 293 \leq x \leq 298\}}} \arrow[d] \\
\overset{4}{\boxed{\text{a material } x \text{ at a temperature } y}} \arrow[r, "y"] \arrow[d, "x"] & \overset{5}{\boxed{\text{a temperature}}} \arrow[r, "\text{in kelvin}"] & \overset{6}{\boxed{\mathbb{R}}} \\
\overset{7}{\boxed{\text{a material}}} & &
\end{tikzcd}$$

- This is clearly a sketch defining room temperature and what it means to be a material at room temperature.
- An underlying typing system could handle the arrow 3 → 6.
- This olog could be referenced and extended by many scientists.
- It represents a fragment of a world-view.
- Moreover, each object and arrow could be "clickable" meaning one could open it up as a table of values.

Another example



1 a company that has hosted me

2 Galois Inc.

3 a company in Portland Oregon

4 a company

5 Portland Oregon

6 (Galois, John Launchbury)

7 a pair (x, y) where x is a company and y is its founder

8 a founder

9 a person

10 John Launchbury

11 a person I play go with

Dreams and speculation

- Imagine a world of people authoring ologs and connecting to each other's ologs, forming a network of knowledge.
- These connections could given by functors (or "spans"), which could only be instantiated if the two ologs are compatible.
- My dream is an "olog network" of people recording their world-views.
    - This is analogous to the semantic web vision (in fact it is possible to convert an olog into a RDF or OWL schema).
    - Politicians could record their views in ologs. It would be interesting to note precise differences between Obama's olog and Palin's olog.
    - Information in the olog network is neither right nor wrong, just "linked into" by others or not.
- Laws should be query-able. Could laws and legislation be recorded in this precise format?

Conclusion

- The purpose of this talk was to refine and extend the “databases are categories” idea from last time.
- With schemas as categories, views, data migration, and integrating with PL become natural.
- Recording world-views as ologs may yield an information network that is instantly compatible with database systems.
- “It’s all categories and functors!” —
  I hope people see category theory as a unified modeling language for information storage, processing, and transfer.

Thanks for hosting me and inviting me to speak!