The Functorial Data Model

Patrick Schultz, David Spivak, Ryan Wisnesky

Department of Mathematics
Massachusetts Institute of Technology

{schultzp, dspivak, wisnesky}@math.mit.edu

Foundational Methods in Computer Science
June 6, 2015
The functorial data model (my name) originated with Rosebrugh et al. in the late 1990s.
  - Schemas are categories, instances are set-valued functors.
  - Spivak proposes using it to solve information integration problems.

I will describe:
  - Rosebrugh's original model (the FDM)
  - How to use the FDM for information integration
  - Extending the FDM towards SQL (FQL)
  - Extending the FDM towards functional programming (FPQL)
  - Conjectures

Sponsored by:
  - ONR grant N000141310260
  - AFOSR grant FA9550-14-1-0031
Category theory

- A category $\mathcal{C}$ consists of
  - a set of objects
  - for all objects $X, Y$ a set $\mathcal{C}(X, Y)$ of arrows
  - for all objects $X$ an arrow $id \in \mathcal{C}(X, X)$
  - for all objects $X, Y, Z$ a function $\circ: \mathcal{C}(Y, Z) \times \mathcal{C}(X, Y) \rightarrow \mathcal{C}(X, Z)$
  - such that $f \circ id = id$ and $id \circ f = f$ and $(f \circ g) \circ h = f \circ (g \circ h)$

- A functor $F: \mathcal{C} \rightarrow \mathcal{D}$ is a function taking objects in $\mathcal{C}$ to objects in $\mathcal{D}$ and arrows $f: X \rightarrow Y$ in $\mathcal{C}$ to arrows $F(f): F(X) \rightarrow F(Y)$ in $\mathcal{D}$ such that $F(id) = id$ and $F(f \circ g) = F(f) \circ F(g)$.

- A category presentation $\mathcal{C}$ consists of
  - a set of nodes
  - for all nodes $X, Y$ a set $\mathcal{C}(X, Y)$ of edges
  - a set of path equations
- A functor presentation $F: \mathcal{C} \rightarrow \mathcal{D}$ is a function taking nodes in $\mathcal{C}$ to nodes in $\mathcal{D}$ and edges $f: X \rightarrow Y$ in $\mathcal{C}$ to paths $F(f): F(X) \rightarrow F(Y)$ in $\mathcal{D}$ such that $\mathcal{C} \vdash p = q$ implies $\mathcal{D} \vdash F(p) = F(q)$. 
The Functorial Data Model

Emp:works $\rightarrow$ manager $\downarrow$ first $\downarrow$ last $\rightarrow$ Dept

Dept:secretary works $\rightarrow$ Emp:manager:works = Emp:works

Dept:secretary:works = Dept

<table>
<thead>
<tr>
<th>Emp</th>
<th>ID</th>
<th>mgr</th>
<th>works</th>
<th>first</th>
<th>last</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>103</td>
<td>q10</td>
<td>Al</td>
<td>Akin</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>102</td>
<td>x02</td>
<td>Bob</td>
<td>Bo</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>103</td>
<td>q10</td>
<td>Carl</td>
<td>Cork</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dept</th>
<th>ID</th>
<th>sec</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>q10</td>
<td>102</td>
<td>CS</td>
<td></td>
</tr>
<tr>
<td>x02</td>
<td>101</td>
<td>Math</td>
<td></td>
</tr>
</tbody>
</table>
Convention

- Omit Dom table, and draw edges \( \bullet \xrightarrow{f} \bullet_{\text{Dom}} \) as \( \bullet - \circ \cdot f : \)

```
manager
Emp
  ↓
  ↓
first
last
Dom
  ↓
  ↓
name

manager
Emp
  ↓
  ↓
first
last
name
```
The Functorial Data Model (abbreviated)

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

<table>
<thead>
<tr>
<th>ID</th>
<th>mgr</th>
<th>works</th>
<th>first</th>
<th>last</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>103</td>
<td>q10</td>
<td>Al</td>
<td>Akin</td>
</tr>
<tr>
<td>102</td>
<td>102</td>
<td>x02</td>
<td>Bob</td>
<td>Bo</td>
</tr>
<tr>
<td>103</td>
<td>103</td>
<td>q10</td>
<td>Carl</td>
<td>Cork</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>sec</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>q10</td>
<td>102</td>
<td>CS</td>
</tr>
<tr>
<td>x02</td>
<td>101</td>
<td>Math</td>
</tr>
</tbody>
</table>
Functorial Data Migration

- A functor $F: S \to T$ is a constraint-respecting mapping:

  $$\begin{align*}
  \text{nodes}(S) & \to \text{nodes}(T) \\
  \text{edges}(S) & \to \text{paths}(T)
  \end{align*}$$

  and it induces three adjoint data migration functors:

  - $\Delta_F: T\text{-inst} \to S\text{-inst}$ (like project)
    \[ S \xrightarrow{F} T \xrightarrow{I} \text{Set} \]
    \[ \Delta_F(I) := I \circ F \]

  - $\Pi_F: S\text{-inst} \to T\text{-inst}$ (like join)
    \[ \Delta_F \dashv \Pi_F \]

  - $\Sigma_F: S\text{-inst} \to T\text{-inst}$ (like outer disjoint union then quotient)
    \[ \Sigma_F \dashv \Delta_F \]
(Project)

\[
\begin{array}{lll}
\text{ID} & \text{Name} & \text{Salary} \\
1 & Alice & $100 \\
2 & Bob & $250 \\
3 & Sue & $300 \\
\end{array}
\]

\[
\begin{array}{ll}
\text{ID} & \text{Age} \\
4 & 20 \\
5 & 20 \\
6 & 30 \\
\end{array}
\]

\[
\begin{array}{llll}
\text{ID} & \text{Name} & \text{Salary} & \text{Age} \\
\hline
a & Alice & $100 & 20 \\
b & Bob & $250 & 20 \\
c & Sue & $300 & 30 \\
\end{array}
\]
Π (Join)

\[ F \]

\[ N \]

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Salary</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Alice</td>
<td>$100</td>
<td>20</td>
</tr>
<tr>
<td>b</td>
<td>Alice</td>
<td>$100</td>
<td>20</td>
</tr>
<tr>
<td>c</td>
<td>Alice</td>
<td>$100</td>
<td>30</td>
</tr>
<tr>
<td>d</td>
<td>Bob</td>
<td>$250</td>
<td>20</td>
</tr>
<tr>
<td>e</td>
<td>Bob</td>
<td>$250</td>
<td>20</td>
</tr>
<tr>
<td>f</td>
<td>Bob</td>
<td>$250</td>
<td>30</td>
</tr>
<tr>
<td>g</td>
<td>Sue</td>
<td>$300</td>
<td>20</td>
</tr>
<tr>
<td>h</td>
<td>Sue</td>
<td>$300</td>
<td>20</td>
</tr>
<tr>
<td>i</td>
<td>Sue</td>
<td>$300</td>
<td>30</td>
</tr>
</tbody>
</table>

N1

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>$100</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>$250</td>
</tr>
<tr>
<td>3</td>
<td>Sue</td>
<td>$300</td>
</tr>
</tbody>
</table>

N2

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Alice</td>
<td>$100</td>
</tr>
<tr>
<td>5</td>
<td>Bob</td>
<td>$250</td>
</tr>
<tr>
<td>6</td>
<td>Sue</td>
<td>$300</td>
</tr>
</tbody>
</table>

Π\_F
\( \Sigma \) (Union)

\[
\begin{array}{c|c|c}
\text{ID} & \text{Name} & \text{Salary} \\
1 & Alice & 100 \\
2 & Bob & 250 \\
3 & Sue & 300 \\
\end{array}
\begin{array}{c|c|c}
\text{ID} & \text{Age} \\
4 & 20 \\
5 & 20 \\
6 & 30 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
\text{ID} & \text{Name} & \text{Salary} & \text{Age} \\
\text{a} & Alice & 100 & \text{null}_1 \\
\text{b} & Bob & 250 & \text{null}_2 \\
\text{c} & Sue & 300 & \text{null}_3 \\
\text{d} & \text{null}_4 & \text{null}_5 & 20 \\
\text{e} & \text{null}_6 & \text{null}_7 & 20 \\
\text{f} & \text{null}_8 & \text{null}_9 & 30 \\
\end{array}
\]
Foreign keys

\[ \Delta_F, \Pi_F, \Sigma_F \]

N1 | N2 | N

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Salary</th>
<th>Age</th>
<th>ID</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>$100</td>
<td>4</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>$250</td>
<td>5</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>Sue</td>
<td>$300</td>
<td>6</td>
<td>6</td>
<td>30</td>
</tr>
</tbody>
</table>

\[ a \quad Alice \quad $100 \quad 20 \]
\[ b \quad Bob \quad $250 \quad 20 \]
\[ c \quad Sue \quad $300 \quad 30 \]
Self-managers

- $\Delta_F$ will copy SelfMgr into Mgr, and put the identity into mgr.

- $\Pi_F$ will migrate into SelfMgr those Emps who are their own mgr.

- $\Sigma_F$ will migrate into SelfMgr representatives of the “management groups” of Emp, i.e. equivalence classes of Emps modulo the equivalence relation generated by mgr.
  - Adjoint functors are only unique up to isomorphism; hence, there are many $\Sigma_F$ functors; each will choose a different representative.
Pivot (Instance ↔ Schema)

Emp

<table>
<thead>
<tr>
<th>ID</th>
<th>mgr</th>
<th>works</th>
<th>first</th>
<th>last</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>103</td>
<td>q10</td>
<td>Al</td>
<td>Akin</td>
</tr>
<tr>
<td>102</td>
<td>102</td>
<td>x02</td>
<td>Bob</td>
<td>Bo</td>
</tr>
<tr>
<td>103</td>
<td>103</td>
<td>q10</td>
<td>Carl</td>
<td>Cork</td>
</tr>
</tbody>
</table>

Dept

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>q10</td>
<td>CS</td>
</tr>
<tr>
<td>x02</td>
<td>Math</td>
</tr>
</tbody>
</table>
Evaluation of the functorial data model

- **Positives:**
  - The category of categories is bi-cartesian closed (model of the STLC).
  - For each category \(C\), the category \(C\)-inst is a topos (model of HOL).
  - Data integrity constraints (path equations) are built-in to schemas.
  - Data migration functors transform entire instances.
  - The FDM is expressive enough for many information integration tasks.
  - Easy to pivot.

- **Negatives:**
  - Data integrity constraints (in schemas) are limited to path equalities.
  - Data migrations lack analog of set-difference.
  - No aggregation.
  - Data migration functors are hard to program directly.
  - Instance isomorphism is too coarse for many integration tasks.
  - Many problems about finitely-presented categories are semi-computable:
    - Path equivalence (required to check functors are constraint-respecting).
    - Generating a category from a presentation (hence the category of finitely-presented categories is not cartesian closed).
The Attribute Problem

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Age</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>20</td>
<td>$100</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>20</td>
<td>$250</td>
</tr>
<tr>
<td>3</td>
<td>Sue</td>
<td>30</td>
<td>$300</td>
</tr>
</tbody>
</table>

≈ (good)

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Age</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Alice</td>
<td>20</td>
<td>$100</td>
</tr>
<tr>
<td>5</td>
<td>Bob</td>
<td>20</td>
<td>$250</td>
</tr>
<tr>
<td>6</td>
<td>Sue</td>
<td>30</td>
<td>$300</td>
</tr>
</tbody>
</table>

≈ (bad)

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Age</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Amy</td>
<td>20</td>
<td>$100</td>
</tr>
<tr>
<td>2</td>
<td>Bill</td>
<td>20</td>
<td>$250</td>
</tr>
<tr>
<td>3</td>
<td>Susan</td>
<td>30</td>
<td>$300</td>
</tr>
</tbody>
</table>
Solving the Attribute Problem

- Mark certain edges to leaf nodes as “attributes”.
  - In this extension, a schema is a category \( C \), a discrete category \( C_0 \), and a functor \( C_0 \to C \). Instances and migrations also generalize.
  - Schemas become special ER (entity-relationship) diagrams.
  - The FDM takes \( C_0 \) to be empty.
  - The example schema below, which was an abbreviation in the FDM, is a bona-fide schema in this extension: attributes are first, last, and name.
# Solved Attribute Problem

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Age</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>20</td>
<td>$100</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>20</td>
<td>$250</td>
</tr>
<tr>
<td>3</td>
<td>Sue</td>
<td>30</td>
<td>$300</td>
</tr>
</tbody>
</table>

≈ (good)

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Age</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Alice</td>
<td>20</td>
<td>$100</td>
</tr>
<tr>
<td>5</td>
<td>Bob</td>
<td>20</td>
<td>$250</td>
</tr>
<tr>
<td>6</td>
<td>Sue</td>
<td>30</td>
<td>$300</td>
</tr>
</tbody>
</table>

≠ (good)

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Age</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Amy</td>
<td>20</td>
<td>$100</td>
</tr>
<tr>
<td>2</td>
<td>Bill</td>
<td>20</td>
<td>$250</td>
</tr>
<tr>
<td>3</td>
<td>Susan</td>
<td>30</td>
<td>$300</td>
</tr>
</tbody>
</table>
FQL - A Functorial Query Language

- The "schemas as ER diagrams" extension to the functorial data model is the basis of FQL.
  - Open-source, graphical IDE available at categoricaldata.net/fql.html.
- FQL translates data migrations of the form

\[ \Sigma_F \circ \Pi_G \circ \Delta_H \]

into SQL and vice versa. Caveats:
  - \( F \) must be a discrete op-fibration (ensures union compatibility).
  - \( G \) must be a surjection on attributes (ensures domain independence).
  - All categories must be finite (ensures computability).
  - FQL \( \mapsto \) SPCU+idgen (sets)
    SPCU (bags) \( \mapsto \) FQL, SPCU (sets) \( \mapsto \) FQL+squash
    selection equality conjunctive and between variables only.
- Theorem: FQL queries are closed under composition.
FQL Demo
FQL evaluation

- Positives:
  - Attributes.
  - Running on SQL enables interoperability and execution speed.
  - Better $\Sigma$ semantics than TGD-only systems (e.g., Clio).

- Negatives:
  - No selection by constants.
  - Relies on fresh ID generation.
  - Cannot change type of data during migration.
  - Attributes not nullable.

- Apply type-theory to FQL to overcome negatives.
FPQL - a functorial programming and query language

- FPQL extends FQL schemas to include edges between attributes.
  - A typing $\Gamma$ is a category with terminal object.
  - A schema $S$ on typing $\Gamma$ is a category extending $\Gamma$ in a special way.
  - An instance $I$ on schema $S$ is a category extending $S$ in a special way.

- Design decision: treat all categories as finitely-presented, and use monoidal Knuth-Bendix to reduce paths.

- FPQL instances are deductive databases, not extensional ones.
  - FPQL allows inconsistent and infinite databases, if desired.
  - FPQL cannot be implemented with SQL, but can borrow implementation techniques from SQL.
Typings

- A typing is a category with terminal object 1:

  ![Diagram]

  $$\text{reverse}.\text{reverse} = \text{id} \quad \text{length} = \text{reverse}.\text{length}$$

- Implicitly includes, for all well-typed edges $e$:

  $$id_1 = !_1 \quad (e : t \to 1) = !_t \quad (e : t \to t').!_{t'} = !_t$$

- Objects are types, arrows are functions.
Schemas

- A schema over a typing $\Gamma$ is a category extending $\Gamma$ with
  - New objects, called *entities*.
  - New arrows from entities to entities, called *foreign keys*.
  - New arrows from entities to types, called *attributes*.
  - New equations.

manager\[\downarrow\]Emp\[\downarrow\] works\[\rightarrow\] Dept

\begin{align*}
\text{manager.works} &= \text{works} & \text{secr.works} &= \text{id}
\end{align*}
Instances

- An instance over a schema $S$ is a category extending $S$ with
  - New edges from 1, called *variables*, such as
    \[
    \text{bill}: 1 \rightarrow \text{Emp} \quad \text{infinity}: 1 \rightarrow \text{Nat}
    \]
  - New equations, such as
    \[
    \text{bill.age} = \text{zero} \quad \text{bill.works.secr.manager} = \text{bill} \quad \text{bill.manager} = \text{bill}
    \]

- Tabular view of instances:

<table>
<thead>
<tr>
<th>Emp</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>manager</td>
<td>works</td>
<td>age</td>
<td>first</td>
</tr>
<tr>
<td>bill</td>
<td>bill</td>
<td>bill.works</td>
<td>zero</td>
<td>bill.first</td>
</tr>
<tr>
<td>bill.works.secr</td>
<td>bill</td>
<td>bill.works</td>
<td>bill.works.secr.age</td>
<td>bill.works.secr.first</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dept</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>secr</td>
<td>name</td>
</tr>
<tr>
<td>bill.works</td>
<td>bill.works.secr</td>
<td>bill.works.name</td>
</tr>
</tbody>
</table>
FPQL Example

Nat: type
zero: Nat
succ: Nat -> Nat

String: type
reverse: String -> String
length: String -> Nat

eq1: reverse.reverse = String

eq2: reverse.length = length

S = schema {

  nodes
  Emp, Dept;

  edges
  age : Emp -> Nat,
  first : Emp -> String,
  name : Dept -> String,
  works : Emp -> Dept,
  secr : Dept -> Emp,
  manager: Emp -> Emp;

  equations
  manager.works = works,
  secr.works = Dept;

}

I = instance {

  variables
  bill : Emp,
  infinity : Nat;

  equations
  bill.age = zero,
  bill.works
    .secr.manager
    = bill;

} : S
Data Migration in FPQL

- When $S$ and $T$ are schemas on typing $\Gamma$, a schema morphism $F: S \rightarrow T$ is a constraint-respecting mapping

\[
\text{nodes}(S) \rightarrow \text{nodes}(T) \quad \text{edges}(S) \rightarrow \text{paths}(T)
\]

that is the identity on $\Gamma$.

- $\Sigma_F$ is defined to be substitution along $F$:

\[v: 1 \rightarrow X \in I \quad \text{implies} \quad v: 1 \rightarrow F(X) \in \Sigma_F(I)\]

- $\Sigma_F \dashv \Delta_F \dashv \Pi_F$

- Migrations $\Sigma_F \circ \Delta_G \circ \Pi_F$, where $F$ is a discrete op-fibration, are closed under composition, and can be written in SQL-like syntax.
Flower Syntax in FPQL

FPQL
select e.first
from Emp as e
where e.manager.manager = e

SQL
select e.first
from Emp as e, Emp as f
where e.manager = f.ID and
    f.manager = e.ID
Set everyone’s manager to their manager’s manager:

```
EmpQuery = {
  from Emp as e
  attributes
    first = e.first
    last = e.last
  edges
    manager = 
      {e=e.manager.manager}:EmpQuery
    works = 
      {d=e.works}:DeptQuery
} : Emp

DeptQuery = {
  from Dept as d
  attributes
    name = d.name
  edges
    secr = {e=d.secr}:EmpQuery
} : Dept
```
Evaluation of FPQL

- Positives
  - Flower syntax
  - Can change type of data
  - Nullable attributes
  - Typings allow functional programming
  - $\Sigma$ is extremely cheap

- Negatives
  - No special support for cartesian closed typings ($\lambda$-calculi)
  - Categories of instances on a fixed schema are not cartesian closed
  - Cannot run on SQL
Conjectures

- An embedded dependency (ED) is a lifting problem.

- The chase is a left Kan extension.

- $\Sigma_F$, $\Delta_F$ and $\Delta_F$, $\Pi_F$ are reverse data exchanges.

- For every data migration $F : S \to T$, there exists an $X$ such that $F$ can be implemented by chasing a set of EDs over $S + T + X$. 
Conclusion

- Initial success using FPQL with NIST
- Deep connections between the FDM and the relational model
- Looking for collaborators

Future work:
- Restrict typings to a particular cartesian closed category, e.g., Java
- FQL flowers : SQL flowers as ? : EDs
- Aggregation
- Generating mappings from matchings
- Entity-resolution
- Algorithms