Relational Algebra by Way of Adjunctions

Jeremy Gibbons
Fritz Henglein
Ralf Hinze
Nicolas Wu
1. Overview

- relational databases in terms of certain *monads* (sets, bags, lists)
- monads support *comprehensions*, providing a *query notation*:
  
  \[
  \begin{align*}
  & (\text{customer.name}, \text{invoice.amount}) \\
  & | \text{customer} \leftarrow \text{customers}, \\
  & \quad \text{invoice} \leftarrow \text{invoices}, \text{invoice.due} \leq \text{today}, \\
  & \quad \text{customer.cid} == \text{invoice.customer}
  \end{align*}
  \]

- monads have nice mathematical foundations via *adjunctions*
- monad structure explains *aggregation, selection, projection*
- less obvious how to explain *join*
2. Galois connections

Relating monotonic functions between two ordered sets:

\[(A, \leq) \perp (B, \subseteq) \quad \text{means} \quad f(b \leq a) \iff b \subseteq g(a)\]

For example,

\[(\mathbb{R}, \leq_{\mathbb{R}}) \perp (\mathbb{Z}, \leq_{\mathbb{Z}}) \quad \text{and} \quad (\mathbb{Z}, \leq) \perp (\mathbb{Z}, \leq)\]

“Change of coordinates” can sometimes simplify reasoning. Eg rhs gives \(n \times k \leq m \iff n \leq m \div k\), and multiplication is easier to reason about than rounding division.
3. Category theory from ordered sets

A category $C$ consists of

- a set* $|C|$ of objects,
- a set* $C(X, Y)$ of arrows $X \to Y$ for each $X, Y : |C|$, 
- identity arrows $id_X : X \to X$ for each $X$
- composition $f \cdot g : X \to Z$ of compatible arrows $g : X \to Y$ and $f : Y \to Z$, 
- such that composition is associative, with identities as units.

Think of a directed graph, with vertices as objects and paths as arrows.

An ordered set $(A, \leq)$ is a degenerate category, with objects $A$ and a unique arrow $a \to b$ iff $a \leq b$.

\[ \cdots \rightarrow -2 \rightarrow -1 \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow \cdots \]

Many categorical concepts are generalisations from ordered sets.

*proviso...
4. Concrete categories

Ordered sets are a **concrete category**: roughly,

- the objects are *sets with additional structure*
- the arrows are *structure-preserving mappings*

For example, category **PoSet** has preordered sets \((A, \leq)\) as objects, and monotonic functions \(h: (A, \leq) \to (B, \sqsubseteq)\) as arrows:

\[
a \leq a' \implies h(a) \sqsubseteq h(a')
\]

For another example, category **CMon** has commutative monoids \((M, \otimes, \varepsilon)\) as objects, and homomorphisms \(h: (M, \otimes, \varepsilon) \to (M', \oplus, \varepsilon')\) as arrows:

\[
\begin{align*}
  h (m \otimes n) &= h m \oplus h n \\
  h \varepsilon &= \varepsilon'
\end{align*}
\]

Trivially, category **Set** has sets (no additional structure) as objects, and total functions as arrows.
5. Functors

Categories are themselves structured objects...

A functor \( F : C \to D \) is an operation on both objects and arrows, preserving the structure: \( F f : F X \to F Y \) when \( f : X \to Y \), and

\[
\begin{align*}
F id_X &= id_{F X} \\
F (f \cdot g) &= F f \cdot F g
\end{align*}
\]

For example, forgetful functor \( U : \text{CMon} \to \text{Set} \):

\[
\begin{align*}
U (M, \otimes, \varepsilon) &= M \\
U (h : (M, \otimes, \varepsilon) \to (M', \oplus, \varepsilon')) &= h : M \to M'
\end{align*}
\]

Conversely, \( \text{Free} : \text{Set} \to \text{CMon} \) generates the free commutative monoid (ie bags) on a set of elements:

\[
\begin{align*}
\text{Free} A &= (\text{Bag} A, \cup, \emptyset) \\
\text{Free} (f : A \to B) &= \text{map} f : \text{Bag} A \to \text{Bag} B
\end{align*}
\]
6. Adjunctions

Adjunctions are the categorical generalisation of Galois connections. Given categories $\mathbf{C}, \mathbf{D}$, and functors $\mathbf{L} : \mathbf{D} \to \mathbf{C}$ and $\mathbf{R} : \mathbf{C} \to \mathbf{D}$, adjunction $\mathbf{C} \dashv \mathbf{D}$ means* $[-] : \mathbf{C}(\mathbf{L} X, Y) \simeq \mathbf{D}(X, \mathbf{R} Y) : [-]$.

The functional programmer's favourite example is given by currying:

$$\text{Set} \dashv \text{Set} \quad \text{with} \quad \text{curry} : \text{Set}(X \times P, Y) \simeq \text{Set}(X, Y^P) : \text{curry}^\circ$$

hence definitions and properties of $\text{apply} = \text{uncurry} \ \text{id}_{Y^P} : Y^P \times P \to Y$. 

*Note: Adjunction definitions and properties are expressed using natural transformations and the concept of adjunctions in category theory.
7. Products and coproducts

\[
\begin{array}{ccc}
\text{Set} & \rightarrow & \text{Set}^2 \\
\rightarrow & \Delta & \rightarrow \\
\Delta & \rightarrow & \rightarrow \text{Set}
\end{array}
\]

with

\[
\begin{align*}
\text{fork} & \colon \text{Set}^2(\Delta A, (B, C)) \cong \text{Set}(A, B \times C) & \colon \text{fork}^\circ \\
\text{junc}^\circ & \colon \text{Set}(A + B, C) \cong \text{Set}^2((A, B), \Delta C) & \colon \text{junc}
\end{align*}
\]

hence

\[
\begin{align*}
dup & = \text{fork } id_{A,A} : \text{Set}(A, A \times A) \\
(fst, snd) & = \text{fork}^\circ id_{B \times C} : \text{Set}^2(\Delta (B, C), (B, C))
\end{align*}
\]

give tupling and projection. Dually for sums and injections. And more generally for any arity—even zero.
8. Free commutative monoids

Free/forgetful adjunction:

\[
\text{CMon} \quad \Downarrow \quad \text{Set} \quad \text{with} \quad [-] : \text{CMon}(\text{Free } A, (M, \otimes, \varepsilon)) \\
\cong \text{Set}(A, U (M, \otimes, \varepsilon)) \quad : \quad [-]
\]

Unit and counit:

\[
single A = [id_{\text{Free } A}] : A \to U (\text{Free } A) \\
\langle M \rangle = [id_M] : \text{Free } (U M) \to M \quad \text{-- for } M = (M, \otimes, \varepsilon)
\]

whence, for \( h : \text{Free } A \to M \) and \( f : A \to U M = M \),

\[
h = \langle M \rangle \cdot \text{Free } f \iff U h \cdot \text{single } A = f
\]

ie 1-to-1 correspondence between (i) homomorphisms from the free commutative monoid (bags) and (ii) their behaviour on singletons.
9. Aggregation

Aggregations are bag homomorphisms:

<table>
<thead>
<tr>
<th>aggregation</th>
<th>monoid</th>
<th>action on singletons</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>((\mathbb{N}, 0, +))</td>
<td>(\cdot a \mapsto 1)</td>
</tr>
<tr>
<td>sum</td>
<td>((\mathbb{R}, 0, +))</td>
<td>(\cdot a \mapsto a)</td>
</tr>
<tr>
<td>max</td>
<td>((\mathbb{Z} \cup {-\infty}, -\infty, \max))</td>
<td>(\cdot a \mapsto a)</td>
</tr>
<tr>
<td>all</td>
<td>((\mathbb{B}, True, \land))</td>
<td>(\cdot a \mapsto a)</td>
</tr>
</tbody>
</table>

Projection \(\pi_i = \text{Bag } i\) is a homomorphism—just functorial action. Selection \(\sigma_p\) is also a homomorphism, to bags, with action

\[
guard : (A \to \mathbb{B}) \to \text{Bag } A \to \text{Bag } A
\]

\[
guard p a = \text{if } p a \text{ then } \cdot a \text{ else } \emptyset
\]

Projection and selection laws follow from homomorphism laws (and from laws of coproducts, since \(\mathbb{B} = 1 + 1\)).
10. Monads

Finite bags form a *monad* \((\text{Bag}, \text{union}, \text{single})\) with

\[
\text{Bag} = \text{U} \cdot \text{Free}
\]

\[
\text{union} : \text{Bag} (\text{Bag} A) \to \text{Bag} A
\]

\[
\text{single} : A \to \text{Bag} A
\]

which justifies the use of comprehension notation

\[
\{ f \ a \ b \mid a \leftarrow x, b \leftarrow g \ a \}
\]

and its equational properties.

In fact, any adjunction \(L \to R\) yields a monad \((T, \mu, \eta)\) on \(D\), where

\[
T = R \cdot L
\]

\[
\mu A = R [\text{id}_A] L : T (T A) \to T A
\]

\[
\eta A = [\text{id}_A] : A \to T A
\]
11. Maps

Database indexes are essentially maps $\text{Map } K V = V^K$. Maps $(-)^K$ from $K$ form a monad (the $\text{Reader}$ monad in Haskell), so arise from an adjunction.

The *laws of exponents* follow from this adjunction, and from those for products and coproducts:

- $\text{Map } 0 V \approx 1$
- $\text{Map } 1 V \approx V$
- $\text{Map } (K_1 + K_2) V \approx \text{Map } K_1 V \times \text{Map } K_2 V$
- $\text{Map } (K_1 \times K_2) V \approx \text{Map } K_1 (\text{Map } K_2 V)$
- $\text{Map } K 1 \approx 1$
- $\text{Map } K (V_1 \times V_2) \approx \text{Map } K V_1 \times \text{Map } K V_2 : \text{merge}$

—ie *merge* is right-to-left half of the latter iso:

$\text{merge} : \text{Map } K V_1 \times \text{Map } K V_2 \rightarrow \text{Map } K (V_1 \times V_2)$
12. Indexing

Relations are in 1-to-1 correspondence with set-valued functions:

\[
\text{Rel} \xrightarrow{\downarrow} \text{Set} \quad \text{where } J \text{ embeds, and } E R : A \to \text{Set } B \text{ for } R : A \sim B.
\]

Moreover, the correspondence remains valid for bags:

\[
\text{index} : \text{Bag } (K \times V) \simeq \text{Map } K (\text{Bag } V)
\]

Together, \textit{index} and \textit{merge} give efficient relational joins:

\[
x f \bowtie_g y = \text{flatten} (\text{Map } K \circ p (\text{merge} (\text{groupBy } f x, \text{groupBy } g y)))
\]

\[
\text{groupBy} : \text{Eq } K \Rightarrow (V \to K) \to \text{Bag } V \to \text{Map } K (\text{Bag } V)
\]

\[
\text{flatten} : \text{Map } K (\text{Bag } V) \to \text{Bag } V
\]

expressible also via \textit{comprehensive comprehensions}
13. Finiteness

A catch:

- being *finite* is important, for aggregations
- begin a *monad* is important, for comprehensions
- *finite bags* form a monad (as above)
- *maps* form a monad, but *finite maps* do not: the unit

\[ \eta \; a = (\lambda k \to a) : A \to \text{Map} \; K \; A \]

generally yields an infinite map.

How to reconcile finiteness of maps with being a monad?
14. Graded monads

Grading (indexing, parametrizing) a monad by a monoid: an indexed family of endofunctors that collectively behave like a monad.

For monoid \( M = (M, \otimes, e) \), the \( M \)-graded monad \( (T, \mu, \eta) \) is a family \( T_m \) of endofunctors indexed by \( m : M \), with

\[
\mu X : T_m (T_n X) \to T_{m \otimes n} X \\
\eta X : X \to T_e X
\]

satisfying the usual laws. These too arise from adjunctions (even though \( T \) itself is not an endofunctor!).

For example, think of finite vectors, indexed by length.

We use the monoid \( (\mathbb{K}^*, +, \langle \rangle) \) of finite sequences of finite key types \( \mathbb{K} \).
15. Query transformations

These can now all be shown by equational reasoning:

\[ \pi_i \cdot \pi_j = \pi_i \quad -- \text{when } i \cdot j = i \]
\[ \sigma_p \cdot \pi_i = \pi_i \cdot \sigma_p \quad -- \text{when } p \cdot i = p \]
\[ \langle M \rangle \cdot \text{Bag } f \cdot \pi_i = \langle M \rangle \cdot \text{Bag } (f \cdot i) \]
\[ \langle M \rangle \cdot \text{Bag } f \cdot \sigma_p = \langle M \rangle \cdot \text{Bag } (\lambda a \rightarrow \text{if } p a \text{ then } f a \text{ else } \varepsilon) \]
\[ \chi f \otimes g y = \text{Bag } \text{swap } (y \otimes_f \chi) \]
\[ (\chi f \otimes g y) \cdot (g \cdot \text{snd}) \otimes_h z = \text{Bag } \text{assoc } (\chi f \otimes (g \cdot \text{fst}) \cdot (y \otimes_h z)) \]
\[ \pi_{i \otimes_j} (\chi f \otimes g y) = \pi_i \chi f' \otimes g' \cdot \pi_j y \quad -- \text{when } f a = g b \iff f' (i a) = g' (j b) \]
\[ \sigma_p (\chi f \otimes g y) = \sigma_q \chi f \otimes g \cdot \sigma_r y \quad -- \text{when } p (a, b) = q a \land r b \]

for monoid \( M = (M, \otimes, \varepsilon) \).
16. Summary

- *monad comprehensions* for database queries
- structure arising from *adjunctions*
- equivalences from *universal properties*
- fitting in *relational joins*, via indexing and graded monads
- calculating *query transformations*


Thanks to EPSRC *Unifying Theories of Generic Programming* for funding.
transformations in relational algebra come from adjunctions