Relational algebra by way of adjunctions
Gibbons, Jeremy; Henglein, Fritz; Hinze, Ralf; Wu, Nicolas

Publication date:
2016

Document version
Early version, also known as pre-print

Citation for published version (APA):
Relational Algebra by Way of Adjunctions

Jeremy Gibbons
(joint work with Fritz Henglein, Ralf Hinze, Nicolas Wu)
DBPL, October 2015
1. Summary

- bulk types (sets, bags, lists) are *monads*
- monads have nice *mathematical foundations via adjunctions*
- monads support *comprehensions*
- comprehension syntax provides a *query notation*

\[
\left\{ (\text{customer.name, invoice.amount}) \mid \text{customer} \leftarrow \text{customers}, \\
\quad \text{invoice} \leftarrow \text{invoices}, \\
\quad \text{customer.cid} = \text{invoice.customer}, \\
\quad \text{invoice.due} \leq \text{today} \right\}
\]

- monad structure explains *selection, projection*
- less obvious how to explain *join*
2. Galois connections

Relating monotonic functions between two ordered sets:

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

For example,

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

“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 $\mathcal{C}$ consists of

- a set* $|\mathcal{C}|$ of objects,
- a set* $\mathcal{C}(X, Y)$ of arrows $X \to Y$ for each $X, Y : |\mathcal{C}|$,
- identity arrows $\text{id}_X : X \to X$ 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 $a \leq b$.

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

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*

Many useful categories are of this form.

For example, the category \( \text{CMon} \) has commutative monoids \((M, \otimes, \epsilon)\) as objects, and homomorphisms \( h : (M, \otimes, \epsilon) \to (M', \oplus, \epsilon') \) as arrows:

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

Trivially, category \( \text{Set} \) has sets 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

$$F \ id_X = id_{F X}$$
$$F (f \cdot g) = F f \cdot F g$$

For example, *forgetful* functor $U : CMon \to Set$:

$$U (M, \otimes, \epsilon) = M$$
$$U (h : (M, \otimes, \epsilon) \to (M', \oplus, \epsilon')) = h : M \to M'$$

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

$$Free A = (Bag A, \cup, \emptyset)$$
$$Free (f : A \to B) = map f : Bag A \to Bag B$$
6. Adjunctions

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

A familiar example is given by currying:

\[
\text{Set} \downarrow \text{Set} \quad \text{with} \quad \text{curry} : \text{Set}(X \times P, Y) \simeq \text{Set}(X, Y^P) : \text{curry}^\circ
\]

hence definitions and properties of \( \text{apply} = \text{uncurry} \ id_{Y^P} : Y^P \times P \to Y \)
7. Products and coproducts

$$\begin{align*}
\text{Set} & \xymatrix{ \ar@/_/[r] & \text{Set}^2 & \ar@/_/[r] & \text{Set}^2 & \ar@/_/[r] & \text{Set} } \\
\Delta & & \ar@/^/[l] & & \Delta & & \ar@/^/[l] & \times
\end{align*}$$

with

\(\text{fork} : \text{Set}^2(\Delta A, (B, C)) \simeq \text{Set}(A, B \times C) : \text{fork}^\circ\)

\(\text{junc}^\circ : \text{Set}(A + B, C) \simeq \text{Set}^2((A, B), \Delta C) : \text{junc}\)

hence

\(\text{dup} = \text{fork id}_{A,A} : \text{Set}(A, A \times A)\)

\((\text{fst, snd}) = \text{fork}^\circ \text{id}_{B \times C} : \text{Set}^2(\Delta(B, C), (B, C))\)

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

Adjunctions often capture embedding/projection pairs:

\[
\begin{align*}
\text{CMon} & \quad \Downarrow \quad \text{Set} \\
\text{Free} & \quad \Rightarrow \quad \text{with} \quad [-] : \text{CMon} \rightarrow \text{Set} \\
\end{align*}
\]

\[
\text{CMon}(\text{Free } A, (M, \otimes, \epsilon)) \cong \text{Set}(A, U(M, \otimes, \epsilon)) : [-]
\]

Unit and counit:

\[
\begin{align*}
\text{single } A & = \lfloor \text{id}_{\text{Free } A} \rfloor : A \rightarrow U(\text{Free } A) \\
\text{reduce } M & = \lfloor \text{id}_M \rfloor : \text{Free } (U M) \rightarrow M \quad \text{-- for } M = (M, \otimes, \epsilon)
\end{align*}
\]

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

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

ie 1-to-1 correspondence between homomorphisms from the free commutative monoid (bags) and 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>({a} \rightarrow 1)</td>
</tr>
<tr>
<td>sum</td>
<td>((\mathbb{R}, 0, +))</td>
<td>({a} \rightarrow a)</td>
</tr>
<tr>
<td>max</td>
<td>((\mathbb{Z}, \text{minBound}, \text{max}))</td>
<td>({a} \rightarrow a)</td>
</tr>
<tr>
<td>min</td>
<td>((\mathbb{Z}, \text{maxBound}, \text{min}))</td>
<td>({a} \rightarrow a)</td>
</tr>
<tr>
<td>all</td>
<td>((\mathbb{B}, \text{True}, \land))</td>
<td>({a} \rightarrow a)</td>
</tr>
<tr>
<td>any</td>
<td>((\mathbb{B}, \text{False}, \lor))</td>
<td>({a} \rightarrow a)</td>
</tr>
</tbody>
</table>

Selection is a homomorphism, to bags, using action

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

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

Laws about selections follow from laws of homomorphisms (and of coproducts, since \(\mathbb{B} = 1 + 1\)).
10. Monads

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

\[
\begin{align*}
\text{Bag} &= U \cdot \text{Free} \\
\text{union} &\colon \text{Bag} (\text{Bag} A) \to \text{Bag} A \\
\text{single} &\colon A \to \text{Bag} A
\end{align*}
\]

which justifies the use of comprehension notation \([f \ a\ b \mid a \leftarrow x, b \leftarrow g\ a\]\. In fact, for any adjunction \(L \dashv R\) between \(\mathbf{C}\) and \(\mathbf{D}\), we get a monad \((T, \mu, \eta)\) on \(\mathbf{D}\), where

\[
\begin{align*}
T &= R \cdot L \\
\mu A &= R \cdot [id_A] \circ L \cdot T \circ (T A) \to T A \\
\eta A &= [id_A] : A \to T A
\end{align*}
\]
11. Maps

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

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

\[
\begin{align*}
\text{Map } 0 V & \simeq 1 \\
\text{Map } 1 V & \simeq V \\
\text{Map } (K_1 + K_2) V & \simeq \text{Map } K_1 V \times \text{Map } K_2 V \\
\text{Map } (K_1 \times K_2) V & \simeq \text{Map } K_1 (\text{Map } K_2 V) \\
\text{Map } K 1 & \simeq 1 \\
\text{Map } K (V_1 \times V_2) & \simeq \text{Map } K V_1 \times \text{Map } K V_2 : \text{merge}
\end{align*}
\]
12. Indexing

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

\[
\begin{align*}
& \rel \xrightarrow{J} \set \xleftarrow{E} \ impose, \text{ and } \ E \ r : A \to \set B \text{ for } R : A \sim B. \\
\text{Moreover, the correspondence remains valid for bags:}
\end{align*}
\]

\[
\begin{align*}
\text{index} : \text{Bag} \ (K \times V) & \simeq \text{Map} \ K \ (\text{Bag} \ V) \\
\text{Together, index and merge give efficient relational joins:}
\end{align*}
\]

\[
\begin{align*}
x_{f \bowtie g} y &= \text{flatten} \ (\text{Map} \ K \ cp \ (\text{merge} \ (\text{groupBy} \ f \ x, \text{groupBy} \ g \ y))) \\
\text{groupBy} : (V \to K) & \to \text{Bag} \ V \to \text{Map} \ K \ (\text{Bag} \ V) \\
\text{flatten} & : \text{Map} \ K \ (\text{Bag} \ V) \to \text{Bag} \ V
\end{align*}
\]
13. Pointed sets and finite maps

Model *finite maps* \( \text{Map}_\ast \) not as partial functions, but *total* functions to a *pointed* codomain \((A, a)\), i.e. a set \(A\) with a distinguished element \(a : A\).

Pointed sets and point-preserving functions form a category \(\text{Set}_\ast\).

There is an adjunction to \(\text{Set}\), via

\[
\begin{array}{ccc}
\text{Set}_\ast & \cong & \text{Set} \\
\downarrow & \Downarrow & \downarrow \\
\text{Maybe} & \rightarrow & \text{U}
\end{array}
\]

where \(\text{Maybe} A \cong 1 + A\) adds a point, and \(\text{U} (A, a) = A\) discards it.

In particular, \((\text{Bag} A, \emptyset)\) is a pointed set. Moreover, \(\text{Bag} f\) is point-preserving, so we get a functor \(\text{Bag}_\ast : \text{Set} \rightarrow \text{Set}_\ast\).

Indexing remains an isomorphism:

\[
\text{index} : \text{Bag}_\ast (K \times V) \cong \text{Map}_\ast K (\text{Bag}_\ast V)
\]
14. Graded monads

A catch: finite maps aren’t a monad, because

$$\eta a = \lambda k \rightarrow a : A \rightarrow \text{Map } K A$$

in general yields an infinite map.

However, finite maps are a graded monad*: for monoid $$(M, \otimes, \epsilon)$$,

$$\mu X : T_m (T_n X) \rightarrow T_{m \otimes n} X$$
$$\eta X : X \rightarrow T_\epsilon X$$

satisfying the usual laws. These too arise from adjunctions*.

We use the monoid $$(\mathbb{K}, \times, 1)$$ of finite key types under product.
15. Conclusions

- *Monad comprehensions* for database queries
- Structure arising from *adjunctions*
- Equivalences from *universal properties*
- Fitting in *relational joins*, via indexing
- To do: calculating *query optimisations*

Thanks to EPSRC *Unifying Theories of Generic Programming* for funding.