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*

```
[ (customer.name, invoice.amount)
| customer ← customers,
  invoice ← invoices,
  customer.cid = invoice.customer,
  invoice.due ≤ today ]
```

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

Relating monotonic functions between two ordered sets:

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

For example,

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

“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 $\mathbf{C}$ consists of

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

$$\ldots \to -2 \to -1 \to 0 \to 1 \to 2 \to \ldots$$

Many categorical concepts are generalisations from ordered sets.

*proviso...
4. Concrete categories

Ordered sets are a \textit{concrete category}: roughly, 

- the objects are \textit{sets with additional structure} 

- the arrows are \textit{structure-preserving mappings} 

Many useful categories are of this form.

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

$$h (m \otimes n) = h m \oplus h n$$
$$h \epsilon = \epsilon'$$

Trivially, category $\mathbf{Set}$ 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 $U : C\text{Mon} \to \text{Set}$:

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

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

$$\text{Free} \ A = (\text{Bag} \ A, \uplus, \emptyset)$$
$$\text{Free} (f : A \to B) = \text{map} \ f : \text{Bag} \ A \to \text{Bag} \ B$$
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

\[
\begin{array}{ccc}
\mathbf{C} & \perp & \mathbf{D} \\
\downarrow \; L & \quad & \; R \downarrow \\
\mathbf{D} & \quad & \mathbf{C}
\end{array}
\]

means* $[-] : \mathbf{C}(L X, Y) \simeq \mathbf{D}(X, R Y) : [-]$.

A familiar example is given by currying:

\[
\begin{array}{ccc}
\mathbf{Set} & \perp & \mathbf{Set} \\
\downarrow \; (-)P & \quad & \; -\times P \downarrow \\
\mathbf{Set} & \quad & \mathbf{Set}
\end{array}
\]

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

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

with

\[ \text{fork} : \mathbf{Set}^2(\Delta A, (B, C)) \simeq \mathbf{Set}(A, B \times C) : \text{fork}^\circ \]
\[ \text{junc}^\circ : \mathbf{Set}(A + B, C) \simeq \mathbf{Set}^2((A, B), \Delta C) : \text{junc} \]

hence

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

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:

\[ \text{CMon} \downarrow \text{Set} \]

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

Unit and counit:

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

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

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

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>$\langle \mathbb{N}, 0, + \rangle$</td>
<td>$\forall a \rightarrow 1$</td>
</tr>
<tr>
<td>sum</td>
<td>$\langle \mathbb{R}, 0, + \rangle$</td>
<td>$\forall a \rightarrow a$</td>
</tr>
<tr>
<td>max</td>
<td>$\langle \mathbb{Z}, \text{minBound}, \text{max} \rangle$</td>
<td>$\forall a \rightarrow a$</td>
</tr>
<tr>
<td>min</td>
<td>$\langle \mathbb{Z}, \text{maxBound}, \text{min} \rangle$</td>
<td>$\forall a \rightarrow a$</td>
</tr>
<tr>
<td>all</td>
<td>$\langle \mathbb{B}, \text{True}, \land \rangle$</td>
<td>$\forall a \rightarrow a$</td>
</tr>
<tr>
<td>any</td>
<td>$\langle \mathbb{B}, \text{False}, \lor \rangle$</td>
<td>$\forall a \rightarrow a$</td>
</tr>
</tbody>
</table>

Selection is a homomorphism, to bags, using action

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

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

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

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

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

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 \( C \) and \( D \), we get a monad \( (T, \mu, \eta) \) on \( D \), where

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

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

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

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

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:

\[
x f \bowtie g y = \text{flatten} (\text{Map } K \text{ cp (merge (groupBy f x, groupBy g y))))
\]

\[
\text{groupBy} : (V \rightarrow K) \rightarrow \text{Bag } V \rightarrow \text{Map } K (\text{Bag } V)
\]

\[
\text{flatten} \quad : \text{Map } K (\text{Bag } V) \rightarrow \text{Bag } V
\]
13. Pointed sets and finite maps

Model *finite maps* $\text{Map}_*$ 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 $\text{Set}_*$. There is an adjunction to $\text{Set}$, via

$$
\begin{align*}
\text{Set}_* & \xleftarrow{\bot} \text{Set} \\
\text{Maybe} & \quad \text{U}
\end{align*}
$$

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

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

Indexing remains an isomorphism:

$$
\text{index} : \text{Bag}_* (K \times V) \simeq \text{Map}_* K (\text{Bag}_* V)
$$
14. Graded monads

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

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

in general yields an infinite map.

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

\[
\begin{align*}
\mu X : & T_m (T_n X) \to T_{m \otimes n} X \\
\eta X : & X \to T_\epsilon X
\end{align*}
\]

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.