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

```plaintext
[ (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:

\[ (A, \leq) \perp (B, \subseteq) \]

means \( f \ b \leq a \iff b \subseteq g \ a \)

For example,

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

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

“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 \( \text{id}_X : X \to X \) 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 \( a \leq b \).

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

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

Many useful categories are of this form.

For example, the category $\text{CMon}$ has commutative monoids $(M, \otimes, \epsilon)$ as objects, and homomorphisms $h : (M, \otimes, \epsilon) \rightarrow (M', \oplus, \epsilon')$ 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* $F : \mathbf{C} \to \mathbf{D}$ 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 $U : \mathbf{CMon} \to \mathbf{Set}$:

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

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

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

Adjunctions are the categorical generalisation of Galois connections.

Given categories $\mathsf{C}, \mathsf{D}$, and functors $L : \mathsf{D} \to \mathsf{C}$ and $R : \mathsf{C} \to \mathsf{D}$, adjunction

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

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

A familiar example is given by currying:

\[
\begin{array}{ccc}
\mathsf{Set} & \downarrow & \mathsf{Set} \\
\mathsf{Set} & \mathsf{Set} & \mathsf{Set} \\
- \times P & \mathsf{(-)P}
\end{array}
\]

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

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

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

with

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

hence

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

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:

\[
\begin{align*}
\text{CMon} & \quad \perp \quad \text{Set} \\
\text{Free} & \quad \downarrow \quad \downarrow \\
\U & \quad \U \\
\end{align*}
\]

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

Unit and counit:

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

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

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

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

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

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

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

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

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

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

which justifies the use of comprehension notation \(\{ f \ a \ b \mid a \leftarrow x, b \leftarrow g \ a \}\).

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

\[
T = R \cdot L
\]

\[
\mu A = R [id_A] L : T (T A) \to T A
\]

\[
\eta A = [id_A] : A \to T A
\]
11. Maps

Database indexes are essentially maps $\text{Map } K V = V^K$. Maps $(-)^K$ from $K$ form a monad (the $\textbf{Reader}$ 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:

\[ \text{Rel} \xrightarrow{\perp} \text{Set} \]

where \( J \) embeds, and \( E R : A \rightarrow \text{Set} B \) 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, \( \text{index} \) and \( \text{merge} \) give efficient relational joins:

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

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

\( \text{flatten} : \text{Map} K (\text{Bag} V) \rightarrow \text{Bag} V \)
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

\[
\begin{array}{ccc}
\text{Set}_* & \overset{\perp}{\longrightarrow} & \text{Set} \\
\downarrow \text{Maybe} & & \downarrow \text{U} \\
\text{Set}_* & \overset{\perp}{\longleftarrow} & \text{Set}
\end{array}
\]

where $\text{Maybe } A \simeq 1 + A$ 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 $\text{Bag}_* : \text{Set} \to \text{Set}_*$.

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 *graded monad*: for monoid $$(M, \otimes, \epsilon)$$,

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

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.