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):
Gibbons, J., Henglein, F., Hinze, R., & Wu, N. (2016). Relational algebra by way of adjunctions. Abstract from 15th International Symposium on Database Programming Languages, Pittsburgh, United States. http://conf.researchr.org/event/dbpl2015/dbpl2015-relational-algebra-by-way-of-adjunctions
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:

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

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; e.g. 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** $X \rightarrow Y$ for each $X, Y : |\mathbf{C}|$, 
- identity arrows $id_X : X \rightarrow X$ for each $X$ 
- composition $f \cdot g : X \rightarrow Z$ of compatible arrows $g : X \rightarrow Y$ and $f : Y \rightarrow Z$, 
- 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 $a \rightarrow b$ 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 \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 \textbf{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 \textbf{Set} has sets as objects, and total functions as arrows.
5. Functors

Categories are themselves structured objects...

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

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

For example, forgetful functor $U : CMon \rightarrow Set$:

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

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

$$Free A = (Bag A, \cup, \emptyset)$$
$$Free (f : A \rightarrow B) = map f : Bag A \rightarrow Bag B$$
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

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

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

A familiar example is given by currying:

\[
\begin{array}{ccc}
\text{Set} & \perp & \text{Set} \\
\uparrow & \ & \downarrow \\
- \times P & \ & (\cdot)^P
\end{array}
\]

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

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

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

with

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

hence

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

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:

\[
\begin{array}{ccc}
\text{CMon} & \perp & \text{Set} \\
\uparrow & & \downarrow \\
\text{Free} & & \U \\
\end{array}
\]

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

Unit and counit:

- \(\text{single } A = [\text{id}_{\text{Free } A}] : A \rightarrow \U (\text{Free } A)\)
- \(\text{reduce } M = [\text{id}_M] : \text{Free } (\U M) \rightarrow M \quad \text{-- for } M = (M, \otimes, \epsilon)\)

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:

| aggregation | monoid       | action on singletons |
|-------------|--------------|----------------------|
| count       | \((\mathbb{N}, 0, +)\) | \(\{a\} \rightarrow 1\) |
| sum         | \((\mathbb{R}, 0, +)\) | \(\{a\} \rightarrow a\) |
| max         | \((\mathbb{Z}, \text{minBound}, \text{max})\) | \(\{a\} \rightarrow a\) |
| min         | \((\mathbb{Z}, \text{maxBound}, \text{min})\) | \(\{a\} \rightarrow a\) |
| all         | \((\mathbb{B}, \text{True}, \wedge)\) | \(\{a\} \rightarrow a\) |
| any         | \((\mathbb{B}, \text{False}, \vee)\) | \(\{a\} \rightarrow a\) |

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

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

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

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

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

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

\[
\begin{array}{c}
\text{Rel} \\
\downarrow \ J \\
\downarrow \ E \\
\text{Set}
\end{array}
\]

where \( J \) embeds, and \( E R : A \to Set B \) 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:

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

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

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

$$\begin{array}{ccc}
\text{Set}_* & \xrightarrow{\perp} & \text{Set} \\
\downarrow \text{Maybe} & & \downarrow \text{U} \\
\end{array}$$

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.