Relational Algebra by Way of Adjunctions

Jeremy Gibbons
(joint work with Fritz Henglein, Ralf Hinze, Nicolas Wu)
S-REPLS#10, Sep 2018

1. Overview

- relational databases in terms of certain *monads* (sets, bags, lists)
- monads support *comprehensions*, providing a *query notation*:

$$[\ (customer.name, invoice.amount)\ |\ customer \leftarrow customers, invoice \leftarrow invoices, customer.cid == invoice.customer, invoice.due \leqslant today\ ]$$

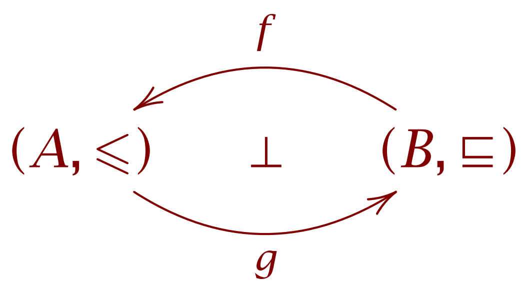
  which are the essence of SQL queries:

$$\mathit{SELECT}\ name, amount\ \mathit{FROM}\ customers, invoices\ \mathit{WHERE}\ cid = customer\ \mathit{AND}\ due \leqslant today$$

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

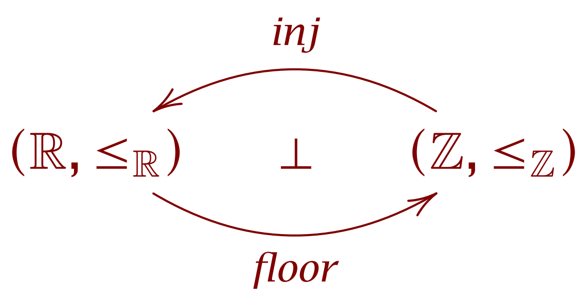
2. Galois connections

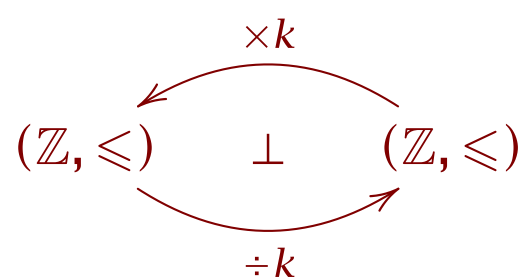
Relating monotonic functions between two ordered sets:



means $f\ b \leqslant a \iff b \sqsubseteq g\ a$

For example,



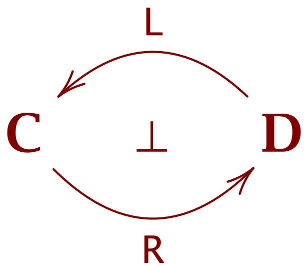


"Change of coordinates" can sometimes simplify reasoning.
Eg rhs gives $n \times k \leqslant m \iff n \leqslant m \div k$, and multiplication is easier to reason about than rounding division.

3. Adjunctions

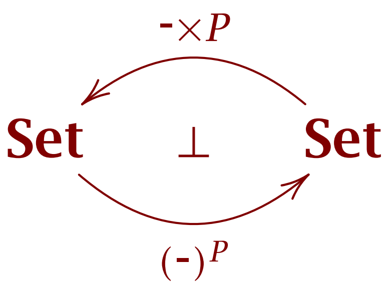
Adjunctions are the categorical generalisation of Galois connections.

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



means* $\lfloor - \rfloor : \mathbf{C}(\mathsf{L}\, X, Y) \simeq \mathbf{D}(X, \mathsf{R}\, Y) : \lceil - \rceil$

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



with $\mathit{curry} : \mathbf{Set}(X \times P, Y) \simeq \mathbf{Set}(X, Y^P) : \mathit{uncurry}$

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

4. Free commutative monoids

Free/forgetful adjunction:

$$\mathbf{CMon} \underset{\mathsf{U}}{\overset{\mathsf{Free}}{\leftrightarrows}} \mathbf{Set} \quad (\bot) \qquad \text{with } \lfloor \text{-} \rfloor : \mathbf{CMon}(\mathsf{Free}\ A, (M, \otimes, \epsilon)) \simeq \mathbf{Set}(A, \mathsf{U}\ (M, \otimes, \epsilon)) : \lceil \text{-} \rceil$$

Unit and counit:

$$\begin{aligned} single\ A &= \lfloor id_{\mathsf{Free}\ A} \rfloor : A \to \mathsf{U}\ (\mathsf{Free}\ A) \\ (\!|\mathsf{M}|\!) &= \lceil id_{\mathsf{M}} \rceil \quad : \mathsf{Free}\ (\mathsf{U}\ \mathsf{M}) \to \mathsf{M} \quad \text{-- for } \mathsf{M} = (M, \otimes, \epsilon) \end{aligned}$$

whence, for $h : \mathsf{Free}\ A \to \mathsf{M}$ and $f : A \to \mathsf{U}\ \mathsf{M} = M$,

$$h = (\!|\mathsf{M}|\!) \cdot \mathsf{Free}\ f \iff \mathsf{U}\ h \cdot single\ A = f$$

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

5. Aggregation

Aggregations are bag homomorphisms:

aggregation	monoid	action on singletons
$count$	$(\mathbb{N}, 0, +)$	$\wr a \int \mapsto 1$
sum	$(\mathbb{R}, 0, +)$	$\wr a \int \mapsto a$
max	$(\mathbb{Z} \cup \{-\infty\}, -\infty, max)$	$\wr a \int \mapsto a$
all	$(\mathbb{B}, True, \wedge)$	$\wr a \int \mapsto a$

Projection $\pi_i = \mathsf{Bag}\ i$ is a homomorphism—just functorial action.
Selection σ_p is also a homomorphism, to bags, with action

$$guard : (A \to \mathbb{B}) \to \mathsf{Bag}\ A \to \mathsf{Bag}\ A$$
$$guard\ p\ a = \mathbf{if}\ p\ a\ \mathbf{then}\ \wr a \int\ \mathbf{else}\ \emptyset$$

Projection and selection laws follow from homomorphism laws.

6. Monads

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

$$\begin{aligned} \mathsf{Bag} &= \mathsf{U} \cdot \mathsf{Free} \\ \mathit{union} &: \mathsf{Bag}\ (\mathsf{Bag}\ A) \to \mathsf{Bag}\ A \\ \mathit{single} &: A \to \mathsf{Bag}\ A \end{aligned}$$

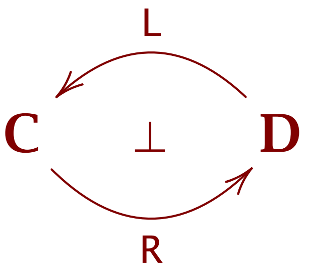
which justifies the use of comprehension notation

$$\wr f\ a\ b \mid a \leftarrow x, b \leftarrow g\ a \wr$$

and its equational properties.

In fact, any adjunction $\mathsf{L} \dashv \mathsf{R}$ yields a monad (T, μ, η) on $\mathbf{D}$, where

$$\begin{aligned} \mathsf{T} &= \mathsf{R} \cdot \mathsf{L} \\ \mu\ A &= \mathsf{R}\ \lceil id_A \rceil\ \mathsf{L} : \mathsf{T}\ (\mathsf{T}\ A) \to \mathsf{T}\ A \\ \eta\ A &= \lfloor id_A \rfloor \quad : A \to \mathsf{T}\ A \end{aligned}$$



7. Maps

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

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

$$
\begin{aligned}
\mathsf{Map}\ 0\ V &\simeq 1 \\
\mathsf{Map}\ 1\ V &\simeq V \\
\mathsf{Map}\ (K_1 + K_2)\ V &\simeq \mathsf{Map}\ K_1\ V \times \mathsf{Map}\ K_2\ V \\
\mathsf{Map}\ (K_1 \times K_2)\ V &\simeq \mathsf{Map}\ K_1\ (\mathsf{Map}\ K_2\ V) \\
\mathsf{Map}\ K\ 1 &\simeq 1 \\
\mathsf{Map}\ K\ (V_1 \times V_2) &\simeq \mathsf{Map}\ K\ V_1 \times \mathsf{Map}\ K\ V_2 : \mathit{merge}
\end{aligned}
$$

8. Indexing

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

$$\mathbf{Rel} \underset{\mathsf{E}}{\overset{\mathsf{J}}{\leftrightarrows}} \mathbf{Set} \quad (\mathsf{J} \dashv \mathsf{E})$$

where J embeds, and $\mathsf{E}\ R : A \to \mathsf{Set}\ B$ for $R : A \sim B$.

Moreover, the correspondence remains valid for bags:

$$\mathit{index} : \mathsf{Bag}\ (K \times V) \simeq \mathsf{Map}\ K\ (\mathsf{Bag}\ V)$$

Together, *index* and *merge* give efficient relational joins:

$$x\ {}_{f}\!\bowtie_{g}\ y = \mathit{flatten}\ (\mathsf{Map}\ K\ \mathit{cp}\ (\mathit{merge}\ (\mathit{groupBy}\ f\ x, \mathit{groupBy}\ g\ y)))$$

$$\mathit{groupBy} : \mathit{Eq}\ K \Rightarrow (V \to K) \to \mathsf{Bag}\ V \to \mathsf{Map}\ K\ (\mathsf{Bag}\ V)$$

$$\mathit{flatten} : \mathsf{Map}\ K\ (\mathsf{Bag}\ V) \to \mathsf{Bag}\ V$$

expressible also via *comprehensive comprehensions*.

9. 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
- *finite maps* do not form a monad: the unit

$$\eta\ a = (\lambda k \rightarrow a) : A \rightarrow \mathsf{Map}\ K\ A$$

  generally yields an infinite map.

How to reconcile finiteness of maps with being a monad?

10. Graded monads

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

For monoid $\mathsf{M} = (M, \otimes, \epsilon)$, the M-graded monad (T, μ, η) is
a family T_m of endofunctors indexed by $m : M$, with

$$\mu\ X : \mathsf{T}_m\ (\mathsf{T}_n\ X) \rightarrow \mathsf{T}_{m \otimes n}\ X$$
$$\eta\ X : X \rightarrow \mathsf{T}_\epsilon\ X$$

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 $\mathbb{K}$.

11. Query transformations

These can now all be shown by equational reasoning:

$$
\begin{array}{lll}
\pi_i \cdot \pi_j & = \pi_i & \text{-- when } i \cdot j = i \\
\sigma_p \cdot \pi_i & = \pi_i \cdot \sigma_p & \text{-- when } p \cdot i = p \\
(\!|\mathsf{M}|\!) \cdot \mathsf{Bag}\ f \cdot \pi_i & = (\!|\mathsf{M}|\!) \cdot \mathsf{Bag}\ (f \cdot i) & \\
(\!|\mathsf{M}|\!) \cdot \mathsf{Bag}\ f \cdot \sigma_p & = (\!|\mathsf{M}|\!) \cdot \mathsf{Bag}\ (\lambda a \rightarrow \mathbf{if}\ p\ a\ \mathbf{then}\ f\ a\ \mathbf{else}\ \epsilon) & \\
x\ {}_f\!\bowtie_g\ y & = \mathsf{Bag}\ \mathit{swap}\ (y\ {}_g\!\bowtie_f\ x) & \\
(x\ {}_f\!\bowtie_g\ y)\ {}_{(g \cdot \mathit{snd})}\!\bowtie_h\ z & = \mathsf{Bag}\ \mathit{assoc}\ (x\ {}_f\!\bowtie_{(g \cdot \mathit{fst})}\ (y\ {}_g\!\bowtie_h\ z)) & \\
\pi_{i \times j}\ (x\ {}_f\!\bowtie_g\ y) & = \pi_i\ x\ {}_{f'}\!\bowtie_{g'}\ \pi_j\ y & \text{-- when } f\ a = g\ b \iff f'\ (i\ a) = g'\ (j\ b) \\
\sigma_p\ (x\ {}_f\!\bowtie_g\ y) & = \sigma_q\ x\ {}_f\!\bowtie_g\ \sigma_r\ y & \text{-- when } p\ (a, b) = q\ a \wedge r\ b
\end{array}
$$

for monoid $\mathsf{M} = (M, \otimes, \epsilon)$.

12. 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*

Paper to appear at ICFP 2018.

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