Relational Algebra by Way of Adjunctions

Jeremy Gibbons
Fritz Henglein
Ralf Hinze
Nicolas Wu

1. Overview

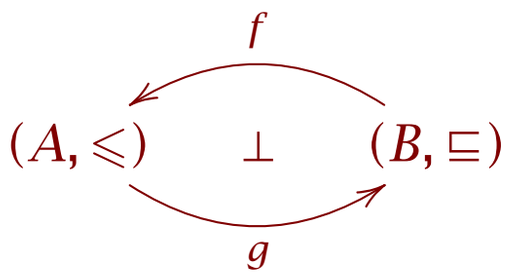
- relational databases in terms of certain *monads* (sets, bags, lists)
- monads support *comprehensions*, providing a *query notation*:

$$
\begin{array}{l}
[\ (customer.name, invoice.amount) \\
\mid customer \leftarrow customers, \\
\quad invoice \leftarrow invoices, invoice.due \leqslant today, \\
\quad customer.cid == invoice.customer\ ]
\end{array}
$$

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

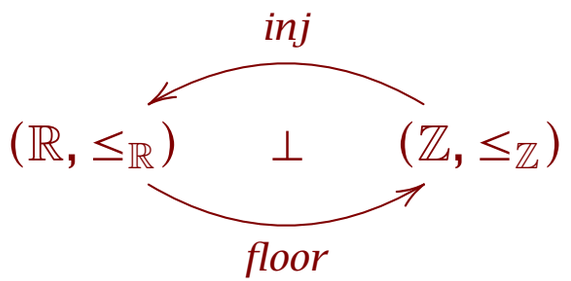
2. Galois connections

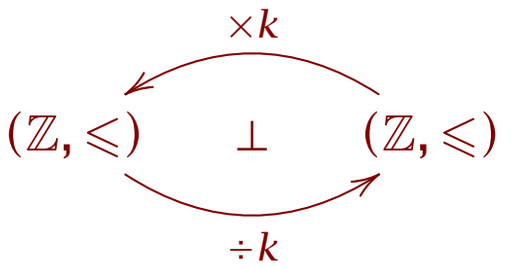
Relating monotonic functions between two ordered sets:



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. 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 $id_X : X \to X$ 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 $(A, \leqslant)$ is a degenerate category, with objects A and a unique arrow $a \to b$ iff $a \leqslant b$.

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*

For example, category **PoSet** has preordered sets $(A, \leqslant)$ as objects, and monotonic functions $h : (A, \leqslant) \to (B, \sqsubseteq)$ as arrows:

$$a \leqslant a' \implies h(a) \sqsubseteq h(a')$$

For another example, category **CMon** has commutative monoids $(M, \otimes, \epsilon)$ as objects, and homomorphisms $h : (M, \otimes, \epsilon) \to (M', \oplus, \epsilon')$ as arrows:

$$\begin{aligned} h\,(m \otimes n) &= h\,m \oplus h\,n \\ h\,\epsilon &= \epsilon' \end{aligned}$$

Trivially, category **Set** has sets (no additional structure) as objects, and total functions as arrows.

5. Functors

Categories are themselves structured objects...

A *functor* $\mathsf{F} : \mathbf{C} \to \mathbf{D}$ is an operation on both objects and arrows, preserving the structure: $\mathsf{F}\ f : \mathsf{F}\ X \to \mathsf{F}\ Y$ when $f : X \to Y$, and

$$\begin{aligned} \mathsf{F}\ id_X &= id_{\mathsf{F}\ X} \\ \mathsf{F}\ (f \cdot g) &= \mathsf{F}\ f \cdot \mathsf{F}\ g \end{aligned}$$

For example, *forgetful* functor $\mathsf{U} : \mathbf{CMon} \to \mathbf{Set}$:

$$\begin{aligned} \mathsf{U}\ (M, \otimes, \epsilon) &= M \\ \mathsf{U}\ (h : (M, \otimes, \epsilon) \to (M', \oplus, \epsilon')) &= h : M \to M' \end{aligned}$$

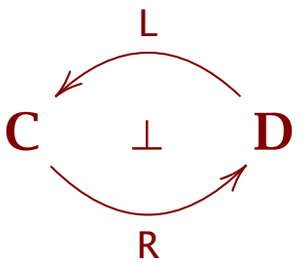
Conversely, $\mathsf{Free} : \mathbf{Set} \to \mathbf{CMon}$ generates the *free* commutative monoid (ie bags) on a set of elements:

$$\begin{aligned} \mathsf{Free}\ A &= (\mathsf{Bag}\ A, \uplus, \varnothing) \\ \mathsf{Free}\ (f : A \to B) &= map\ f : \mathsf{Bag}\ A \to \mathsf{Bag}\ B \end{aligned}$$

6. Adjunctions

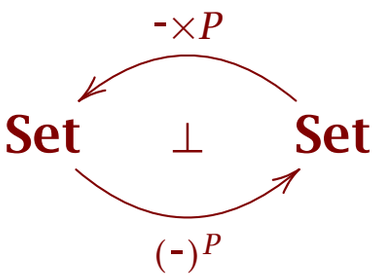
Adjunctions are the categorical generalisation of Galois connections.

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

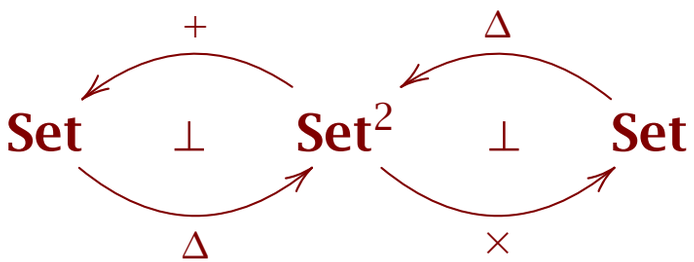
The functional programmer's favourite example is given by *currying*:



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

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

7. Products and coproducts



with

$$\begin{aligned} fork &: \mathbf{Set}^2(\Delta A, (B, C)) \simeq \mathbf{Set}(A, B \times C) &&: fork^\circ \\ junc^\circ &: \mathbf{Set}(A + B, C) \simeq \mathbf{Set}^2((A, B), \Delta C) &&: junc \end{aligned}$$

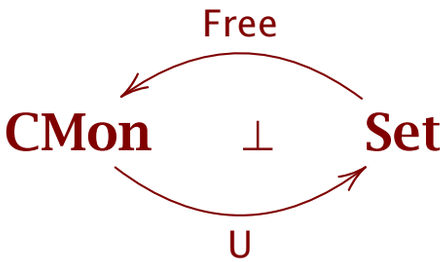
hence

$$\begin{aligned} dup &= fork\ id_{A,A} &&: \mathbf{Set}(A, A \times A) \\ (fst, snd) &= fork^\circ\ id_{B \times C} &&: \mathbf{Set}^2(\Delta(B, C), (B, C)) \end{aligned}$$

give tupling and projection. Dually for sums and injections.
And more generally for any arity—even zero.

8. Free commutative monoids

Free/forgetful adjunction:



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

Unit and counit:

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

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

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

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

9. Aggregation

Aggregations are bag homomorphisms:

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

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

$$guard : (A \to \mathbb{B}) \to \mathsf{Bag}\ A \to \mathsf{Bag}\ A$$
$$guard\ p\ a = \textbf{if}\ p\ a\ \textbf{then}\ ⟅a⟆\ \textbf{else}\ \varnothing$$

Projection and selection laws follow from homomorphism laws
(and from laws of coproducts, since $\mathbb{B} = 1 + 1$).

10. Monads

Finite bags form a *monad* (Bag, *union*, *single*) with

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

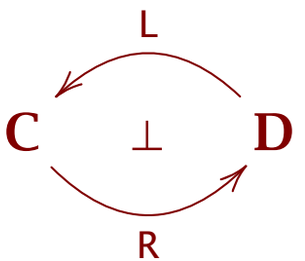
which justifies the use of comprehension notation

$$\wr f\ a\ b \vee a \leftarrow x, b \leftarrow g\ a \int$$

and its equational properties.

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

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



11. 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:

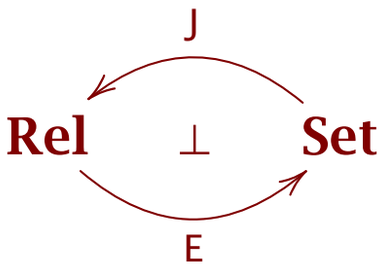
$$\begin{aligned}
\mathsf{Map}\ 0\ V &\simeq 1 \\
\mathsf{Map}\ 1\ V &\simeq V \\
\mathsf{Map}\ (K_1 + K_2)\ V &\simeq \mathsf{Map}\ K_1\ V \times \mathsf{Map}\ K_2\ V \\
\mathsf{Map}\ (K_1 \times K_2)\ V &\simeq \mathsf{Map}\ K_1\ (\mathsf{Map}\ K_2\ V) \\
\mathsf{Map}\ K\ 1 &\simeq 1 \\
\mathsf{Map}\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

—ie *merge* is right-to-left half of the latter iso:

$$\mathit{merge} : \mathsf{Map}\ K\ V_1 \times \mathsf{Map}\ K\ V_2 \to \mathsf{Map}\ K\ (V_1 \times V_2)$$

12. Indexing

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



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

Moreover, the correspondence remains valid for bags:

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

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

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

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

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

expressible also via *comprehensive comprehensions*

13. 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, but *finite maps* do not: the unit

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

  generally yields an infinite map.

How to reconcile finiteness of maps with being a monad?

14. 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

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

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 $\mathbb{K}$.

15. Query transformations

These can now all be shown by equational reasoning:

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

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

16. 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.

transformations in relational algebra come from adjunctions