# Relational Algebra by Way of Adjunctions

*Jeremy Gibbons*
*(joint work with Fritz Henglein, Ralf Hinze, Nicolas Wu)*
*DBPL, October 2015*

# 1. Summary

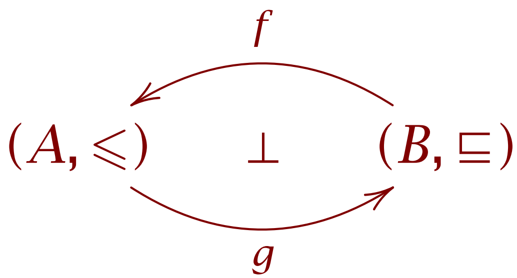
- bulk types (sets, bags, lists) are *monads*
- monads have nice *mathematical foundations* via *adjunctions*
- monads support *comprehensions*
- comprehension syntax provides a *query* notation

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

- monad structure explains *selection*, *projection*
- less obvious how to explain *join*

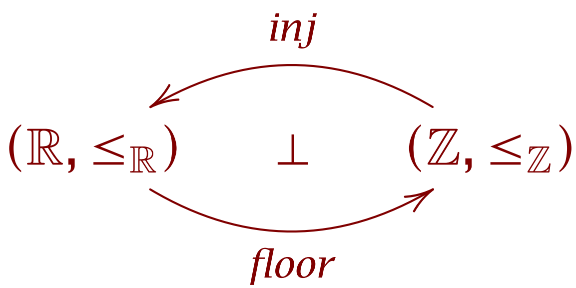
# 2. Galois connections

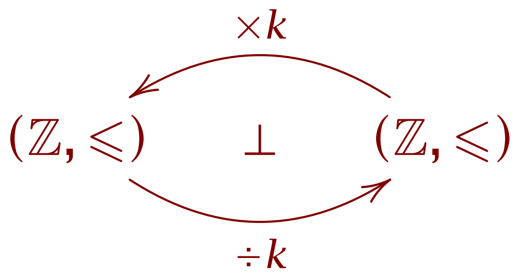
Relating monotonic functions between two ordered sets:




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 $a \leqslant b$.

$$\cdots \longrightarrow -2 \longrightarrow -1 \longrightarrow 0 \longrightarrow 1 \longrightarrow 2 \longrightarrow \cdots$$

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 **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 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 $\mathsf{U} : \mathbf{CMon} \to \mathbf{Set}$:

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

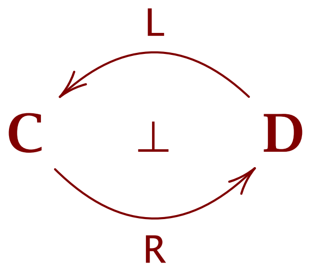
Conversely, $\mathsf{Free} : \mathbf{Set} \to \mathbf{CMon}$ generates the *free* commutative monoid (ie bags) on a set of elements:

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

# 6. Adjunctions

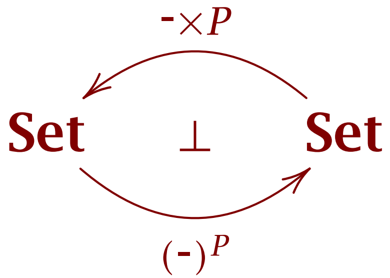
*Adjunctions* are the categorical generalisation of Galois connections.

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

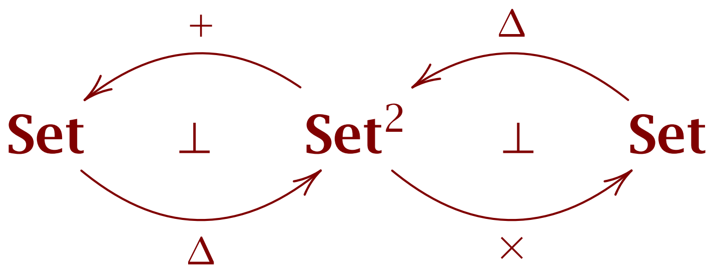
A familiar example is given by *currying*:




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

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

# 7. Products and coproducts




with

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

hence

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

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:

$$\mathbf{CMon} \underset{\mathsf{U}}{\overset{\mathsf{Free}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathbf{Set}$$

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:

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

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

$$h = \mathit{reduce}\ \mathsf{M} \cdot \mathsf{Free}\ f \iff \mathsf{U}\ h \cdot \mathit{single}\ A = f$$

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* | $(\mathbb{N}, 0, +)$ | $\wr a \int \mapsto 1$ |
| *sum* | $(\mathbb{R}, 0, +)$ | $\wr a \int \mapsto a$ |
| *max* | $(\mathbb{Z}, \mathit{minBound}, \mathit{max})$ | $\wr a \int \mapsto a$ |
| *min* | $(\mathbb{Z}, \mathit{maxBound}, \mathit{min})$ | $\wr a \int \mapsto a$ |
| *all* | $(\mathbb{B}, \mathit{True}, \wedge)$ | $\wr a \int \mapsto a$ |
| *any* | $(\mathbb{B}, \mathit{False}, \vee)$ | $\wr a \int \mapsto a$ |

Selection is a homomorphism, to bags, using action

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

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

# 10. Monads

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

In fact, for any adjunction $\mathsf{L} \dashv \mathsf{R}$ between $\mathbf{C}$ and $\mathbf{D}$, we get a monad $(\mathsf{T}, \mu, \eta)$ on $\mathbf{D}$, where

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

# 11. Maps

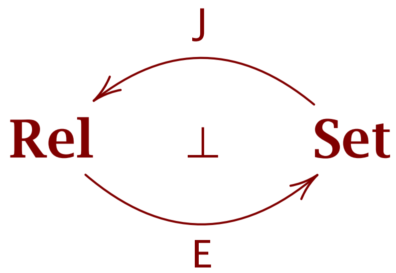
Database indexes are essentially maps $\mathsf{Map}\ K\ V = V^K$. Maps $(\text{-})^K$ from $K$ form a monad (the *Reader* monad in Haskell), so arise from an adjunction.

The *laws of exponents* arise from this adjunction, and from those for products and coproducts:

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

# 12. Indexing

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




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:

$$x \;{}_f{\bowtie}_g\; y = \mathit{flatten}\;(\mathsf{Map}\;K\;\mathit{cp}\;(\mathit{merge}\;(\mathit{groupBy}\;f\;x, \mathit{groupBy}\;g\;y)))$$

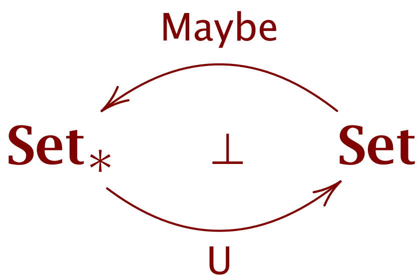
$$\mathit{groupBy} : (V \to K) \to \mathsf{Bag}\;V \to \mathsf{Map}\;K\;(\mathsf{Bag}\;V)$$

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

# 13. Pointed sets and finite maps

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




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

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

Indexing remains an isomorphism:

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

# 14. Graded monads

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

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

in general yields an infinite map.

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

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

satisfying the usual laws. These too arise from adjunctions*.

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.