Comprehending Ringads

Jeremy Gibbons

WadlerFest, April 2016

1. Comprehensions

- ZF axiom schema of specification:

$$\{x^2 \mid x \in \mathit{Nat} \wedge x < 10 \wedge x \text{ is even}\}$$

- SETL set-formers:

$$\{x * x : x \;\mathbf{in}\; \{0 \,..\, 9\} \mid x \;\mathbf{mod}\; 2 = 0\}$$

- Eindhoven Quantifier Notation:

$$(x : 0 \leqslant x < 10 \wedge x \text{ is even} : x^2)$$

- Haskell (NPL, Python, ...) list comprehensions:

$$[x \mathbin{\hat{}} 2 \mid x \leftarrow [0 \,..\, 9], \mathit{even}\; x]$$

2. Comprehending monads (Wadler, 1992)

Structure is that of a *monad with zero* $(\mathsf{T}, \mathit{return}, \mathit{mult}, \varnothing)$:

$$
\begin{array}{ll}
\mathit{mult} :: \mathsf{T}\ (\mathsf{T}\ a) \to \mathsf{T}\ a & \mathit{mult}\ (\mathit{mult}\ \mathit{xss}) = \mathit{mult}\ (\mathit{fmap}\ \mathit{mult}\ \mathit{xss}) \\
\mathit{return} :: a \to \mathsf{T}\ a & \mathit{mult}\ (\mathit{return}\ x) = x \\
\mathit{mzero} :: \mathsf{T}\ a & \mathit{mult}\ (\mathit{fmap}\ \mathit{return}\ x) = x \\
 & \mathit{mult}\ \varnothing = \varnothing
\end{array}
$$

Comprehensions can then be used for any monad-with-zero:

$$
\begin{array}{ll}
\mathcal{D}\,[\,e \mid \ \,] & = \mathit{return}\ e \\
\mathcal{D}\,[\,e \mid p \leftarrow e', Q\,] & = e' \ggg\!= \lambda p \to \mathcal{D}\,[\,e \mid Q\,] \\
\mathcal{D}\,[\,e \mid e', Q\,] & = \mathit{guard}\ e' \ggg\!= \lambda() \to \mathcal{D}\,[\,e \mid Q\,] \\
\mathcal{D}\,[\,e \mid \mathbf{let}\ d, Q\,] & = \mathbf{let}\ d\ \mathbf{in}\ \mathcal{D}\,[\,e \mid Q\,]
\end{array}
$$

(where $x \ggg\!= k = \mathit{mult}\ (\mathit{fmap}\ k\ x)$ and $\mathit{guard}\ b = \mathbf{if}\ b\ \mathbf{then}\ \mathit{return}\ ()\ \mathbf{else}\ \varnothing$).

Hence monad comprehensions for *sets*, *bags*, *(sub-)distributions*, *exceptions*...

3. Collection monads

Finite *collection* types are monads. But the operations of a monad-with-zero cannot introduce multiplicity; need also

$$(\uplus) :: \mathsf{T}\ a \to \mathsf{T}\ a \to \mathsf{T}\ a$$

such that

$$\begin{aligned} \mathit{mult}\ (\mathit{xs} \uplus \mathit{ys}) &= (\mathit{mult}\ \mathit{xs}) \uplus (\mathit{mult}\ \mathit{ys}) \\ x \uplus \emptyset &= x \\ \emptyset \uplus y &= y \end{aligned}$$

Sets, *bags*, *sub-distributions* are collection monads; but *exceptions* are not.

Eg the *Boom Hierarchy*:		
	trees	
	lists	$\uplus$ is associative
	bags	... and commutative
	sets	... and idempotent

4. Aggregations

Well-behaved operations h over collections: $count$, sum, $some$, ...

$$\begin{aligned} h\ (return\ a) &= a \\ h\ (mult\ xs) &= h\ (fmap\ h\ xs) \end{aligned}$$

—the *algebras* for the monad T.

Define $\oplus$ and ε by

$$\begin{aligned} a \oplus b &= h\ (return\ a \uplus return\ b) \\ \varepsilon &= h\ \emptyset \end{aligned}$$

Then

$$h\ (x \uplus y) = h\ x \oplus h\ y$$

Moreover, $\oplus$ and ε satisfy whatever laws $\uplus$ and $\emptyset$ do:
ε is the unit of $\oplus$; $\oplus$ is associative if $\uplus$ is; etc (at least on the range of h).

5. Comprehending queries

Wadler & Trinder (1991) argued for comprehensions as a query notation: Given input tables

$$\begin{aligned}&\mathit{customers} :: \mathit{Bag}\ (\mathit{CID}, \mathit{Name}, \mathit{Address})\\&\mathit{invoices} :: \mathit{Bag}\ (\mathit{IID}, \mathit{CID}, \mathit{Amount}, \mathit{Date})\end{aligned}$$

then

$$\begin{aligned}\mathit{overdueInvoices} = [\ &(c.\mathit{name}, c.\mathit{address}, i.\mathit{amount})\\ \mid\ &c \leftarrow \mathit{customers},\\ &i \leftarrow \mathit{invoices}, i.\mathit{due} < \mathit{today},\\ &c.\mathit{cid} == i.\mathit{customer}\ ]\end{aligned}$$

Works similarly in any collection monad, not just bags.

An influential observation in the DBPL community: basis of languages such as Buneman's *Kleisli*, Microsoft *LINQ*, Wadler's *Links*, as well as querying for objects (*OQL*) and XML (*XQuery*).

6. The problem with joins

The comprehension yields a terrible query plan!
Constructs entire cartesian product, then discards most of it:

$$\begin{array}{l} \mathit{fmap}\ (\lambda(c,i) \rightarrow (c.\mathit{name}, c.\mathit{address}, i.\mathit{amount})\ (\\ \quad \mathit{filter}\ (\lambda(c,i) \rightarrow c.\mathit{cid} == i.\mathit{customer})\ (\\ \quad\quad \mathit{filter}\ (\lambda(c,i) \rightarrow i.\mathit{due} < \mathit{today})\ (\\ \quad\quad\quad \mathit{cp}\ \mathit{customers}\ \mathit{invoices}))) \end{array}$$

Better to group by customer identifier, then handle groups separately:

$$\begin{array}{l} \mathit{fmap}\ (\mathit{fmap}\ (\lambda c \rightarrow (c.\mathit{name}, c.\mathit{address})) \times \mathit{fmap}\ (\lambda i \rightarrow i.\mathit{amount}))\ (\\ \quad \mathit{fmap}\ (\mathit{id} \times \mathit{filter}\ (\lambda i \rightarrow i.\mathit{due} < \mathit{today}))\ (\\ \quad\quad \mathit{merge}\ (\mathit{indexBy}\ \mathit{cid}\ \mathit{customers}, \mathit{indexBy}\ \mathit{customer}\ \mathit{invoices}))) \end{array}$$

(where *indexBy* partitions, and *merge* pairs on common index).
But this doesn't correspond to anything expressible in comprehensions.

7. Comprehensive comprehensions

Parallel ('zip') comprehensions (Clean 1.0, 1995):

$$[\,(x,y) \mid x \leftarrow [\,1,2,3\,] \mid y \leftarrow [\,4,5,6\,]\,] = [\,(1,4),(2,5),(3,6)\,]$$

'Order by' and 'group by' (Wadler & Peyton Jones, 2007):

$$\begin{array}{l}
[\ (\mathit{the}\ \mathit{dept}, \mathit{sum}\ \mathit{salary}) \\
\mid (\mathit{name}, \mathit{dept}, \mathit{salary}) \leftarrow \mathit{employees} \\
,\ \textbf{then group by}\ \mathit{dept}\ \textbf{using}\ \mathit{groupWith} \\
,\ \textbf{then}\ \mathit{sortWith}\ \textbf{by}\ \mathit{sum}\ \mathit{salary}\,]
\end{array}$$

(NB **group by** rebinds the variables *salary* etc bound earlier!)
Initially just for lists, but also generalizable (Giorgidze et al., 2011):

$$\begin{array}{ll}
\mathit{mzip}_{\mathsf{T}} & :: \mathsf{T}\ a \to \mathsf{T}\ b \to \mathsf{T}\ (a,b) \\
\mathit{mgroupWith}_{\mathsf{T},\mathsf{U},\mathsf{F}} & :: \mathit{Eq}\ b \Rightarrow (a \to b) \to \mathsf{T}\ a \to \mathsf{U}\ (\mathsf{F}\ a)
\end{array}$$

(Note heterogeneous type: T, U should be monads, F a functor.)

8. Solving the problem with (equi-)joins

Maps-to-bags form a monad-with-zero—roughly:

$$\textbf{type}\ \mathit{Map}\ k\ v \quad = k \to v$$
$$\textbf{type}\ \mathit{Table}\ k\ v = \mathit{Map}\ k\ (\mathit{Bag}\ v)$$

Now define

$$\mathit{merge} \quad :: (\mathit{Table}\ k\ v, \mathit{Table}\ k\ w) \to \mathit{Table}\ k\ (v, w)$$
$$\mathit{merge}\ (f, g) = \lambda k \to \mathit{cp}\ (f\ k)\ (g\ k)$$

$$\mathit{indexBy} :: \mathit{Eq}\ k \Rightarrow (v \to k) \to \mathit{Bag}\ v \to \mathit{Table}\ k\ v$$
$$\mathit{indexBy}\ f\ \mathit{xs}\ k = \mathit{filter}\ (\lambda v \to f\ v == k)\ \mathit{xs}$$

and use *merge* for zipping, *indexBy* for grouping.

With care, *indexBy* can be evaluated in linear time.

Now represent query as:

$$
\begin{aligned}
&\mathit{overdueInvoices} :: \mathit{Map\ Int\ (Name, Address, Bag\ Amount)} \\
&\mathit{overdueInvoices} = [\ (\mathit{the\ name}, \mathit{the\ addr}, \mathit{amount}) \\
&\qquad |\ (\mathit{cid}, \mathit{name}, \mathit{addr}) \leftarrow \mathit{customers} \\
&\qquad ,\ \mathbf{then\ group\ by}\ \mathit{cid}\ \mathbf{using}\ \mathit{indexBy} \\
&\qquad |\ (\mathit{iid}, \mathit{customer}, \mathit{amount}, \mathit{due}) \leftarrow \mathit{invoices} \\
&\qquad ,\ \mathit{due} < \mathit{today} \\
&\qquad ,\ \mathbf{then\ group\ by}\ \mathit{customer}\ \mathbf{using}\ \mathit{indexBy}\ ]
\end{aligned}
$$

Avoids expanding the whole cartesian product.

9. Finite maps

A catch:

- need *monads*, for comprehensions
- need *Map*s, for indexing
- need *finite collections*, for aggregation
- but *finite maps* don't form a monad (no *return*)

Solution?

10. Graded monads (Katsumata et al, 2016)

Monad $(\mathsf{T}, \mathit{return}, \mathit{mult})$ has endofunctor $\mathsf{T} : \mathbf{C} \to \mathbf{C}$, polymorphic functions

$$
\begin{aligned}
\mathit{return} &:: a \to \mathsf{T}\ a \\
\mathit{mult} &:: \mathsf{T}\ (\mathsf{T}\ a) \to \mathsf{T}\ a
\end{aligned}
$$

satisfying certain laws.

M-graded monad $(\mathsf{T}, \mathit{return}, \mathit{mult})$ for monoid $(M, \varepsilon, \cdot)$ has (non-endo-)functor $\mathsf{T} : M \to [\mathbf{C}, \mathbf{C}]$ and

$$
\begin{aligned}
\mathit{return} &:: a \to \mathsf{T}\ \varepsilon\ a \\
\mathit{mult} &:: \mathsf{T}\ m\ (\mathsf{T}\ n\ a) \to \mathsf{T}\ (m \cdot n)\ a
\end{aligned}
$$

with same laws. (Eg for collecting *effects*; think also of *vectors*.)

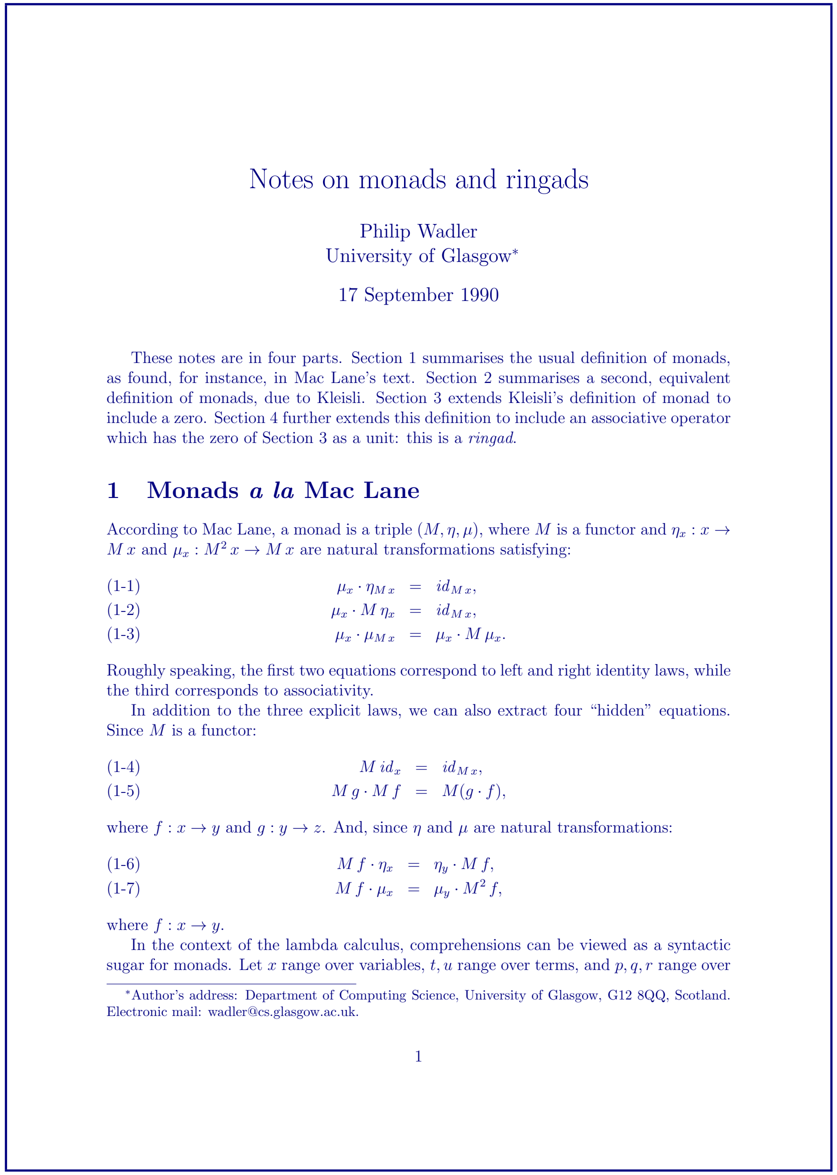
We use $\mathsf{T} = \mathit{Table}$, with monoid $(K, \langle\,\rangle, +\!\!+)$ of finite type sequences.
Not an endofunctor, but there is still a story involving adjunctions.

11. Ringads (Wadler, 1990)

Wadler called collection monads *ringads*.

Ringads are (roughly) *right near-semirings* in the right near-semiringy category of endofunctors under composition and product.

Wadler's note cited in numerous papers from the 1990s (with varying degrees of accuracy), but long thought lost…



Notes on monads and ringads

Philip Wadler
University of Glasgow*

17 September 1990

These notes are in four parts. Section 1 summarises the usual definition of monads, as found, for instance, in Mac Lane's text. Section 2 summarises a second, equivalent definition of monads, due to Kleisli. Section 3 extends Kleisli's definition of monad to include a zero. Section 4 further extends this definition to include an associative operator which has the zero of Section 3 as a unit: this is a *ringad*.

1 Monads *a la* Mac Lane

According to Mac Lane, a monad is a triple (M, η, μ), where M is a functor and $\eta_x : x \to M\,x$ and $\mu_x : M^2\,x \to M\,x$ are natural transformations satisfying:

(1-1) $\mu_x \cdot \eta_{M\,x} = id_{M\,x}$,
(1-2) $\mu_x \cdot M\,\eta_x = id_{M\,x}$,
(1-3) $\mu_x \cdot \mu_{M\,x} = \mu_x \cdot M\,\mu_x$.

Roughly speaking, the first two equations correspond to left and right identity laws, while the third corresponds to associativity.

In addition to the three explicit laws, we can also extract four "hidden" equations. Since M is a functor:

(1-4) $M\,id_x = id_{M\,x}$,
(1-5) $M\,g \cdot M\,f = M(g \cdot f)$,

where $f : x \to y$ and $g : y \to z$. And, since η and μ are natural transformations:

(1-6) $M\,f \cdot \eta_x = \eta_y \cdot M\,f$,
(1-7) $M\,f \cdot \mu_x = \mu_y \cdot M^2\,f$,

where $f : x \to y$.

In the context of the lambda calculus, comprehensions can be viewed as a syntactic sugar for monads. Let x range over variables, t, u range over terms, and p, q, r range over

*Author's address: Department of Computing Science, University of Glasgow, G12 8QQ, Scotland. Electronic mail: wadler@cs.glasgow.ac.uk.

1

12. Conclusions: Comprehending Wadler

- list *comprehensions*
- *monads* for functional programming
- *monad comprehensions* and **do** notation
- comprehensions for *queries*
- *comprehensive* comprehensions
- *graded monads* for the marriage with effects

- thank you, Phil!