Monads from Comonads
Comonads from Monads

Ralf Hinze

Computing Laboratory, University of Oxford
Wolfson Building, Parks Road, Oxford, OX1 3QD, England
ralf.hinze@comlab.ox.ac.uk
http://www.comlab.ox.ac.uk/ralf.hinze/

March 2011

Buy one get one free!

A common form of sales promotion (BOGOF).

1 Monads

Monads, a success story.

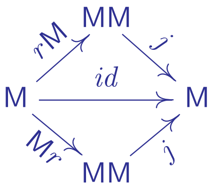
$$A \to \mathsf{M}\, B$$

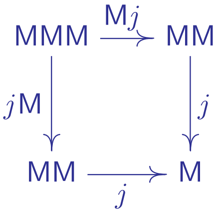
A monad consists of a functor M and natural transformations

$$r : \mathsf{I} \to \mathsf{M}$$
$$j : \mathsf{MM} \to \mathsf{M}$$

The two operations have to go together:

$$j \cdot r\mathsf{M} = id_{\mathsf{M}}$$
$$j \cdot \mathsf{M}r = id_{\mathsf{M}}$$
$$j \cdot \mathsf{M}j = j \cdot j\mathsf{M}$$





1 Comonads

Comonads, not exactly a success story.

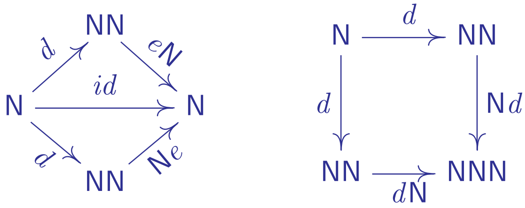
$$\mathsf{N}\,A \to B$$

A comonad consists of a functor N and natural transformations

$$e : \mathsf{N} \to \mathsf{I}$$
$$d : \mathsf{N} \to \mathsf{NN}$$

The two operations have to go together:

$$\mathsf{N}e \cdot d = id_{\mathsf{N}}$$
$$e\mathsf{N} \cdot d = id_{\mathsf{N}}$$
$$\mathsf{N}d \cdot d = d\mathsf{N} \cdot d$$



The simplest of all: the *product comonad*.

$$\mathsf{N} = - \times X$$
$$e = outl$$
$$d = id \vartriangle outr$$

Why has the product comonad not taken off?

2 Adjunctions

- One of the most beautiful constructions in mathematics.
- They allow us to transfer a problem to another domain.
- They provide a framework for program transformations.

Let $\mathscr{C}$ and $\mathscr{D}$ be categories. The functors $\mathsf{L} : \mathscr{C} \leftarrow \mathscr{D}$ and $\mathsf{R} : \mathscr{C} \rightarrow \mathscr{D}$ are *adjoint*, $\mathsf{L} \dashv \mathsf{R}$,

$$\mathscr{C} \;\underset{\mathsf{R}}{\overset{\mathsf{L}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}}\; \mathscr{D}$$

if and only if there is a bijection between the hom-sets

$$\mathscr{C}(\mathsf{L}\,A, B) \cong \mathscr{D}(A, \mathsf{R}\,B)$$

that is natural both in A and B.

The witness of the isomorphism is called the *left adjunct*. It allows us to trade L in the source for R in the target of an arrow. Its inverse is the *right adjunct*.

Perhaps the best-known example of an adjunction is currying.

$$\mathscr{C} \underset{(-)^X}{\overset{- \times X}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C} \qquad\qquad \Lambda : \mathscr{C}(A \times X, B) \cong \mathscr{C}(A, B^X)$$

The left adjunct Λ is also called *curry* and the right adjunct Λ° is also called *uncurry*.

$$\mathscr{C} \underset{(-)^X}{\overset{- \times X}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C} \qquad \Lambda : \mathscr{C}(A \times X, B) \cong \mathscr{C}(A, B^X)$$

The images of the identity are function application and the *return* of the state monad.

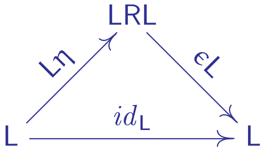
$$app = \Lambda^{\circ}\, id : \mathscr{C}(B^X \times X, B)$$
$$r = \Lambda\, id : \mathscr{C}(A, (A \times X)^X)$$

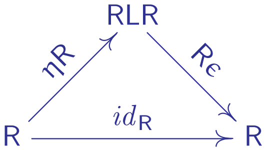
The images of the identity are the counit and the unit of the adjunction.

$$\epsilon : \mathsf{LR} \to \mathsf{I}$$
$$\eta : \mathsf{I} \to \mathsf{RL}$$

An alternative definition of adjunctions builds solely on these units, which have to satisfy

$$\epsilon\mathsf{L} \cdot \mathsf{L}\eta = id_{\mathsf{L}}$$
$$\mathsf{R}\epsilon \cdot \eta\mathsf{R} = id_{\mathsf{R}}$$





2 Comonads and monads

Every adjunction $\mathsf{L} \dashv \mathsf{R}$ induces a comonad and a monad.

$$\mathsf{N} = \mathsf{L}\mathsf{R} \qquad \mathsf{M} = \mathsf{R}\mathsf{L}$$

$$e = \epsilon \qquad r = \eta$$

$$d = \mathsf{L}\eta\mathsf{R} \qquad j = \mathsf{R}\epsilon\mathsf{L}$$

The curry adjunction induces the *state monad*.

$$\mathsf{M}\,A = (A \times X)^X$$
$$r = \Lambda\, id$$
$$j = \Lambda\,(app \cdot app)$$

The monad supports stateful computations, where the state X is threaded through a program.

The curry adjunction induces the *costate comonad*.

$$\mathsf{N}\,A = A^X \times X$$
$$e = app$$
$$d = (\Lambda\, id) \times X$$

The context can be seen as a store A^X together with a memory location X, a focus of interest.

3 Transforming natural transformations

- Adjunctions provide a framework for program transformations.
- All the operations we have encountered so far are natural transformations.
- To deal effectively with those we develop a little theory of 'natural transformation transformers'.

3 Post-composition

Every adjunction $\mathsf{L} \dashv \mathsf{R}$ gives rise to an adjunction $\mathsf{L}- \dashv \mathsf{R}-$ between functor categories.

$$\mathscr{C} \underset{\mathsf{R}}{\overset{\mathsf{L}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{D} \quad \text{then} \quad \mathscr{C}^{\mathscr{E}} \underset{\mathsf{R}-}{\overset{\mathsf{L}-}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{D}^{\mathscr{E}}$$

$$\mathscr{C}^{\mathscr{E}}(\mathsf{LF}, \mathsf{G}) \cong \mathscr{D}^{\mathscr{E}}(\mathsf{F}, \mathsf{RG})$$

We write $\lfloor - \rfloor$ for the lifted left adjunct and $\lfloor - \rfloor^{\circ}$ for its inverse.

$$\frac{\alpha : \mathsf{LF} \to \mathsf{G}}{\lfloor \alpha \rfloor : \mathsf{F} \to \mathsf{RG}} \qquad \frac{\beta : \mathsf{F} \to \mathsf{RG}}{\lfloor \beta \rfloor^{\circ} : \mathsf{LF} \to \mathsf{G}}$$

The lifted adjuncts can be defined in terms of the units of the underlying adjunction:

$$\lfloor \alpha \rfloor = \mathsf{R}\alpha \cdot \eta\mathsf{F} \qquad \lfloor \beta \rfloor^{\circ} = \epsilon\mathsf{G} \cdot \mathsf{L}\beta$$

3 Pre-composition

Post-composition dualizes to pre-composition. Consequently, every adjunction $\mathsf{L} \dashv \mathsf{R}$ also induces an adjunction $-\mathsf{R} \dashv -\mathsf{L}$.

$$\mathscr{C} \underset{\mathsf{R}}{\overset{\mathsf{L}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{D} \qquad \text{then} \qquad \mathscr{E}^{\mathscr{D}} \underset{-\mathsf{L}}{\overset{-\mathsf{R}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{E}^{\mathscr{C}}$$

$$\mathscr{E}^{\mathscr{D}}(\mathsf{FR}, \mathsf{G}) \cong \mathscr{E}^{\mathscr{C}}(\mathsf{F}, \mathsf{GL})$$

We write $\lceil - \rceil^{\circ}$ for the lifted left adjunct and $\lceil - \rceil$ for its inverse.

$$\frac{\beta : \mathsf{F}\mathsf{R} \rightarrow \mathsf{G}}{\lceil \beta \rceil^{\circ} : \mathsf{F} \rightarrow \mathsf{G}\mathsf{L}} \qquad \frac{\alpha : \mathsf{F} \rightarrow \mathsf{G}\mathsf{L}}{\lceil \alpha \rceil : \mathsf{F}\mathsf{R} \rightarrow \mathsf{G}}$$

Again, the lifted adjuncts can be defined in terms of the units of the underlying adjunction:

$$\lceil \beta \rceil^{\circ} = \beta \mathsf{L} \cdot \mathsf{F}\eta \qquad \lceil \alpha \rceil = \mathsf{G}\epsilon \cdot \alpha \mathsf{R}$$

An aide-mémoire: $\lfloor - \rfloor$ turns an L in the source to an R in the target, while $\lceil - \rceil$ turns an L in the target to an R in the source.

3 Transformation transformers

If we combine $\lfloor - \rfloor$ and $\lceil - \rceil$, we can send natural transformations of type $\mathsf{LF} \to \mathsf{GL}$ to transformations of type $\mathsf{FR} \to \mathsf{RG}$.

The order in which we apply the adjuncts does not matter.

$$\lfloor \lceil \alpha \rceil \rfloor = \lceil \lfloor \alpha \rfloor \rceil \qquad \lceil \lfloor \beta \rfloor^{\circ} \rceil^{\circ} = \lfloor \lceil \beta \rceil^{\circ} \rfloor^{\circ}$$

If we assume that L and R are endofunctors, then we can nest $\lfloor - \rfloor$ and $\lceil - \rceil$ arbitrarily deep.

$$\frac{\alpha : \mathsf{L}^m \mathsf{F} \to \mathsf{G} \mathsf{L}^n}{\lceil \lfloor \alpha \rfloor^m \rceil^n : \mathsf{R}^n \mathsf{F} \to \mathsf{G} \mathsf{R}^m}$$

An aide-mémoire: the number of $\lfloor$s corresponds to the number of Ls in the source, and the number of $\lceil$s corresponds to the number of Ls in the target.

4 Monads from comonads

- Assume that a left adjoint is at the same time a comonad.
- Then its right adjoint is a monad!
- Dually, the left adjoint of a monad is a comonad.

The ‘transformation transformers’ allow us to systematically turn the comonadic operations into monadic ones and vice versa.

$$r = \lfloor e \rfloor : \mathsf{I} \to \mathsf{R} \qquad e = \lfloor r \rfloor^{\circ} : \mathsf{L} \to \mathsf{I}$$

$$j = \lfloor \lceil \lceil d \rceil \rceil \rfloor : \mathsf{RR} \to \mathsf{R} \qquad d = \lceil \lceil \lfloor j \rfloor^{\circ} \rceil^{\circ} \rceil^{\circ} : \mathsf{L} \to \mathsf{LL}$$

The curry adjunction, provides an example, where the left adjoint $\mathsf{L} = - \times X$ is also a comonad.

$$e = \mathit{outl}$$
$$d = \mathit{id} \vartriangle \mathit{outr}$$

Consequently, L's right adjoint $\mathsf{R} = (-)^X$ is a monad with operations

$$r = \lfloor \mathit{outl} \rfloor = \Lambda\, \mathit{outl}$$
$$j = \lfloor \lceil \lceil \mathit{id} \vartriangle \mathit{outr} \rceil \rceil \rfloor = \Lambda\, (\mathit{app} \cdot (\mathit{app} \vartriangle \mathit{outr}))$$

The resulting structure is known as the *reader monad*.

$$A \times X \to B \cong A \to B^X$$

Every (co)monad comes equipped with additional operations. The product comonad might provide a getter and an update operation:

$$\begin{aligned} get &= outr : \mathsf{L} \to \Delta_X \\ update\,(f : X \to X) &= id \times f : \mathsf{L} \to \mathsf{L} \end{aligned}$$

where Δ_X is the constant functor.

The transforms of *get* and *update* correspond to operations called *ask* and *local* in the Haskell monad transformer library.

$$\begin{aligned} ask &= \lfloor outr \rfloor = \Lambda\, outr : \mathsf{I} \to \mathsf{R}\,\Delta_X \\ local\,(f : X \to X) &= \lfloor \lceil id \times f \rceil \rfloor = \Lambda\,(app \cdot (id \times f)) : \mathsf{R} \to \mathsf{R} \end{aligned}$$

4 Proof

We have to show that the comonadic laws imply the monadic laws and vice versa. (It is sufficient to concentrate on natural transformations of type $\mathsf{L}^m \to \mathsf{L}^n$ and $\mathsf{R}^n \to \mathsf{R}^m$.)

The transformers enjoy functorial properties:

$$\lfloor \lceil id_{\mathsf{L}^n} \rceil^n \rfloor^n = id_{\mathsf{R}^n}$$

$$\lceil \lfloor \beta \cdot \alpha \rfloor^k \rceil^n = \lceil \lfloor \alpha \rfloor^k \rceil^m \cdot \lceil \lfloor \beta \rfloor^m \rceil^n$$

$$\lfloor \lceil \mathsf{L}\alpha \rceil^{n+1} \rfloor^{m+1} = \lfloor \lceil \alpha \rceil^n \rfloor^m \mathsf{R}$$

For reference, we call the last one *flip law*.

The first comonadic unit law is equivalent to the first monadic unit law:

$$
\begin{aligned}
& \mathsf{L}e \cdot d = id_{\mathsf{L}} \\
\Longleftrightarrow \quad & \{ \text{ inverses } \} \\
& \lfloor \lceil \mathsf{L}e \cdot d \rceil \rfloor = \lfloor \lceil id_{\mathsf{L}} \rceil \rfloor \\
\Longleftrightarrow \quad & \{ \text{ preservation of composition and identity } \} \\
& \lfloor \lceil \lceil d \rceil \rceil \rfloor \cdot \lfloor \lfloor \lceil \mathsf{L}e \rceil \rfloor \rfloor = id_{\mathsf{R}} \\
\Longleftrightarrow \quad & \{ \text{ flip law } \} \\
& \lfloor \lceil \lceil d \rceil \rceil \rfloor \cdot \lfloor e \rfloor \mathsf{R} = id_{\mathsf{R}} \\
\Longleftrightarrow \quad & \{ \text{ definition of } j \text{ and definition of } r \} \\
& j \cdot r\mathsf{R} = id_{\mathsf{R}}
\end{aligned}
$$

5 The wrong way round

- Does the translation also work if the *left* adjoint is simultaneously a *monad*?
- The transformers happily take the monadic operations to comonadic ones.
- However, monadic programs of type $A \to \mathsf{L}\,B$ are not in one-to-one correspondence to comonadic programs of type $\mathsf{R}\,A \to B$.

If X is a monoid with operations $[\,] : X$ and $(+\!\!+) : X \times X \to X$, then $\mathsf{L} = - \times X$ also has the structure of a monad.

$$r\ a = (a, [\,])$$
$$j\ ((a, x_1), x_2) = (a, x_1 +\!\!+ x_2)$$

(For simplicity, we assume that we are working in **Set**.) This instance is known as the "write to a monoid" monad or simply the *writer monad*.

Its right adjoint $\mathsf{R} = (-)^X$ is indeed a comonad, lovingly called the "read from a monoid" comonad.

$$e\,f = f\,[\,]$$
$$d\,f = \lambda\,x_1\,.\,\lambda\,x_2\,.\,f\,(x_1 \mathbin{+\!\!+} x_2)$$

However, we cannot translate the accompanying infrastructure of the writer monad. Consider the *write* operation.

$$
\begin{aligned}
\mathit{write} &: X \to \mathsf{L}\, X \\
\mathit{write}\, x &= (x, x)
\end{aligned}
$$

The L is on the wrong side of the arrow, *write* is not natural, so it has no counterpart in the comonadic world.

6 Summary

- Monads: effectful computations.
- Comonads: computations in context.
- Adjunctions: a theory of program transformations.

  $$\mathscr{C}(\mathsf{L}\,A, B) \cong \mathscr{D}(A, \mathsf{R}\,B)$$

  If L and R are endofunctors:

  $$\mathsf{L}^m \to \mathsf{L}^n \cong \mathsf{R}^n \to \mathsf{R}^m$$

- If L is a comonad, then R is a monad. Furthermore, comonadic programs are in one-to-one correspondence to monadic programs: $\mathsf{L}\,A \to B \cong A \to \mathsf{R}\,B$.
- If L is a monad, then R is a comonad. However, monadic programs are *not* in one-to-one correspondence to comonadic programs: $A \to \mathsf{L}\,B \not\cong \mathsf{R}\,A \to B$.

6 Summary continued

The curry adjunction

$$\mathscr{C} \underset{(-)^X}{\overset{- \times X}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{C} \qquad\qquad \Lambda : \mathscr{C}(A \times X, B) \cong \mathscr{C}(A, B^X)$$

explains:

- state monad,
- costate comonad,
- product comonad,
- reader monad,
- "write to a monoid" monad or writer monad,
- "read from a monoid" comonad.

7 Post- and pre-composition

Functors can be composed, written simply using juxtaposition KF. The operation $\mathsf{K}-$, post-composing a functor K, is itself functorial:

$$\begin{aligned} \mathsf{K}- &: \mathscr{D}^{\mathscr{C}} \to \mathscr{E}^{\mathscr{C}} \\ (\mathsf{K}-)\,\mathsf{F} &= \mathsf{KF} \\ (\mathsf{K}-)\,\alpha &= \mathsf{K}\alpha \end{aligned}$$

where $(\mathsf{K}\alpha)\,A = \mathsf{K}\,(\alpha\,A)$.

Post-composition dualizes to pre-composition:

$$\begin{aligned} -\mathsf{E} &: \mathscr{D}^{\mathscr{C}} \to \mathscr{D}^{\mathscr{B}} \\ (-\mathsf{E})\,\mathsf{F} &= \mathsf{FE} \\ (-\mathsf{E})\,\alpha &= \alpha\mathsf{E} \end{aligned}$$

where $(\alpha\mathsf{E})\,A = \alpha\,(\mathsf{E}\,A)$.