Theoretical Pearl
Monads from Comonads, Comonads from Monads
An Exercise in Program Transformation

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/

1 Introduction

Shall I structure my programs using comonads or monads? Functional programmers have embraced monads but they have not fallen in love with comonads. This is despite the fact that the simplest example of a (co)monadic structure is a comonad, the product comonad. This pearl explains why the product comonad has not taken off. Along the way, we develop a little theory of program transformations. But we are skipping ahead, let us review comonads and monads first.

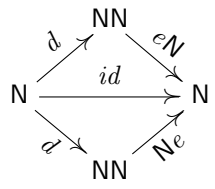
2 Comonads and monads

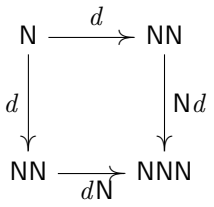
A comonad consists of three pieces of data: an endofunctor N and natural transformations

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

It is helpful to think of a comonad as a mechanism that supports computations in context. A comonadic program is an arrow of type $\mathsf{N}\,A \to B$, where the comonad is wrapped around the source. The operations that come with a comonad manipulate this context: e (short for *e*xtract) discards the context, d (short for *d*uplicate) creates two copies of it. 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 first two coherence properties, the unit laws, express that doubling a context and then discarding one of the copies gives the original context. The third coherence property, the associative law, requires that the two ways of creating three copies of a context are equivalent. (As an aside, $\mathsf{N}e$ denotes the composition of the functor N with the natural transformation e. In case you need to brush up a bit on your category theory, Appendix A contains a refresher.)

An instance of the abstract concept is the costate comonad $\mathsf{N}\,A = A^X \times X$ defined

$$e = app \ ,$$
$$d = (\Lambda\, id) \times X \ .$$

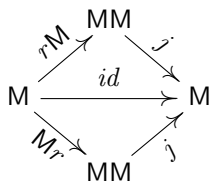
The context can be seen as a store A^X together with a memory location X, a focus of interest. Extracting the data in the focus is implemented simply by function application. We defer a proof that the definitions above define a comonad until later when this will fall out as a by-product.

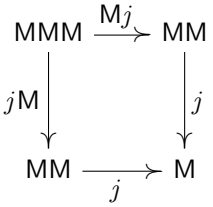
Comonads dualize to monads. For reference, let us spell out the details. A monad consists of an endofunctor M and natural transformations

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

Think of a monad as a mechanism that supports effectful computations. A monadic program is an arrow of type $A \to \mathsf{M}\,B$, where the monad is wrapped around the target. The operations that come with a monad organise effects: r (short for $r$eturn) creates a pure computation, j (short for $j$oin) merges two layers of effects. Like for comonads, the two operations have to go together:

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





The unit laws state that merging a pure with a potentially effectful computation gives the effectful computation. The associative law expresses that the two ways of merging three layers of effects are equivalent.

A popular instance of the abstract concept is the state monad $\mathsf{M}\,A = (A \times X)^X$ defined

$$r = \Lambda\, id \ ,$$
$$j = \Lambda\,(app \cdot app) \ .$$

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

The definitions of the costate comonad and the state monad expose striking similarities: they both involve a product $- \times X$ and an exponential $(-)^X$. This is, of course, not a coincidence. The two structures are intimately related as they both arise out of the same adjunction, a concept we study next.

3 Adjunctions

Adjunctions are among the most beautiful constructions in mathematics. Loosely speaking, an adjunction allows us to transfer a problem from one domain to another, where the problem is possibly simpler to solve. Our interest in adjunctions stems from the fact that they provide a general framework for program transformations. Here is the categorical description.

Let $\mathscr{C}$ and $\mathscr{D}$ be categories. The functors $\mathsf{L} : \mathscr{C} \leftarrow \mathscr{D}$ and $\mathsf{R} : \mathscr{C} \to \mathscr{D}$ 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 functor L is said to be a *left adjoint* for R, while R is L's *right adjoint*. The function witnessing 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*.

Because of the naturality requirement the adjuncts are fully determined by the image of a single arrow, the identity. These two images are called the counit $\epsilon : \mathsf{LR} \to \mathsf{I}$ and the unit $\eta : \mathsf{I} \to \mathsf{RL}$ of the adjunction. In fact, an alternative definition of adjunctions builds solely on these units, which have to satisfy

$$\epsilon \mathsf{L} \cdot \mathsf{L}\eta = id_{\mathsf{L}} \ , \qquad (1) \qquad \mathsf{R}\epsilon \cdot \eta \mathsf{R} = id_{\mathsf{R}} \ . \qquad (2)$$

These so-called triangle identities are equivalent to the requirement that the left adjunct is inverse to the right adjunct.

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

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

The existence of this adjunction is one of the requirements for cartesian closure. The left adjunct Λ is also called *curry*, hence the name curry adjunction. The counit of the adjunction is application, its unit is $\Lambda\, id$.

Every adjunction $\mathsf{L} \dashv \mathsf{R}$ induces a comonad $\mathsf{N} = \mathsf{LR}$ and a monad $\mathsf{M} = \mathsf{RL}$. Their operations have simple implementations in terms of the units.

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

The unit laws are consequences of the triangle identities (1) and (2). The associative law follows from the coherence property of horizontal composition (35).

The curry adjunction induces the (co)state (co)monad. The original definitions of the operations are obtained from the generic ones by plugging in the definitions of $\mathsf{L}f = f \times X$, $\mathsf{R}\,g = \Lambda\,(g \cdot app)$, $\epsilon = app$, and $\eta = \Lambda\, id$.

All the operations we have encountered so far are natural transformations. In fact, this pearl is 100% natural—the forthcoming proofs only involve natural transformations. To deal effectively with those we lift adjunctions to functor categories, developing a little theory of 'transformation transformers' in the remainder of this section.

3.1 Lifting adjunctions

Every adjunction $\mathsf{L} \dashv \mathsf{R}$ gives rise to an adjunction $\mathsf{L}- \dashv \mathsf{R}-$ between functor categories, where $\mathsf{L}-$ is the higher-order functor that takes F to LF and α to $\mathsf{L}\alpha$ (see Appendix A).

$$\mathscr{C} \underset{\mathsf{R}}{\overset{\mathsf{L}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{D} \qquad \text{then} \qquad \mathscr{C}^{\mathscr{E}} \underset{\mathsf{R}-}{\overset{\mathsf{L}-}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{D}^{\mathscr{E}} \qquad \mathscr{C}^{\mathscr{E}}(\mathsf{LF}, \mathsf{G}) \cong \mathscr{D}^{\mathscr{E}}(\mathsf{F}, \mathsf{RG})$$

The lifted isomorphism establishes a bijection between natural transformations and is itself natural in F and G. We write $\lfloor - \rfloor$ for the *lifted* left adjunct and $\lfloor - \rfloor^\circ$ for its inverse. We have noted before that an adjunction can be defined either in terms of adjuncts or in terms of units. To avoid the need for disambiguation we refer to the underlying adjunction only in terms of the units and to the lifted adjunction only via the adjuncts. The lifted adjuncts can be defined in terms of the units of the underlying adjunction as follows:

$$\lfloor \alpha : \mathsf{LF} \to \mathsf{G} \rfloor = \mathsf{R}\alpha \cdot \eta\mathsf{F} \ , \qquad (3) \qquad \lfloor \beta : \mathsf{F} \to \mathsf{RG} \rfloor^\circ = \epsilon\mathsf{G} \cdot \mathsf{L}\beta \ . \qquad (4)$$

As a warm-up for the forthcoming calculations, let us prove that $\lfloor - \rfloor^\circ$ is left-inverse to $\lfloor - \rfloor$.

$$\begin{aligned}
& \lfloor \lfloor \alpha \rfloor \rfloor^\circ \\
= & \quad \{ \text{ definition of } \lfloor - \rfloor \ (3) \ \} \\
& \lfloor \mathsf{R}\alpha \cdot \eta\mathsf{F} \rfloor^\circ \\
= & \quad \{ \text{ definition of } \lfloor - \rfloor^\circ \ (4) \ \} \\
& \epsilon\mathsf{G} \cdot \mathsf{L}(\mathsf{R}\alpha \cdot \eta\mathsf{F}) \\
= & \quad \{ \ \mathsf{L}- \text{ functor } (27) \ \} \\
& \epsilon\mathsf{G} \cdot \mathsf{LR}\alpha \cdot \mathsf{L}\eta\mathsf{F}
\end{aligned}$$

$$
\begin{aligned}
&= \quad \{ \text{ horizontal composition (35): } \epsilon : \mathsf{LR} \to \mathsf{I} \text{ and } \alpha : \mathsf{LF} \to \mathsf{G} \ \} \\
&\quad \alpha \cdot \epsilon \mathsf{LF} \cdot \mathsf{L}\eta\mathsf{F} \\
&= \quad \{ \ -\mathsf{F} \text{ functor (29) } \} \\
&\quad \alpha \cdot (\epsilon \mathsf{L} \cdot \mathsf{L}\eta)\mathsf{F} \\
&= \quad \{ \text{ triangle identity (1) } \} \\
&\quad \alpha \cdot \mathit{id}_{\mathsf{L}}\mathsf{F} \\
&= \quad \{ \ -\mathsf{F} \text{ functor (28) } \} \\
&\quad \alpha \cdot \mathit{id}_{\mathsf{LF}} \\
&= \quad \{ \text{ identity } \} \\
&\quad \alpha
\end{aligned}
$$

That $\lfloor - \rfloor^\circ$ is right-inverse to $\lfloor - \rfloor$ follows by duality. All of the following statements come in pairs, where one identity is dual to the other. Consequently, we only have to prove one of them. Also the subsequent proofs will be less detailed, omitting obvious steps such as manipulating the identity.

The naturality properties of the adjuncts give rise to fusion laws, which allow us to move natural transformations in and out of the brackets.

$$
\begin{aligned}
\lfloor \alpha \rfloor \cdot \beta &= \lfloor \alpha \cdot \mathsf{L}\beta \rfloor && (5) \qquad \lfloor \alpha \rfloor^\circ \cdot \mathsf{L}\beta &= \lfloor \alpha \cdot \beta \rfloor^\circ \\
\mathsf{R}\alpha \cdot \lfloor \beta \rfloor &= \lfloor \alpha \cdot \beta \rfloor && (6) \qquad \alpha \cdot \lfloor \beta \rfloor^\circ &= \lfloor \mathsf{R}\alpha \cdot \beta \rfloor^\circ
\end{aligned}
$$

A direct consequence of the definitions (3) and (4) are the shift laws, which allow us to do the same with functors.

$$
\lfloor \alpha \rfloor \mathsf{H} = \lfloor \alpha \mathsf{H} \rfloor \qquad (7) \qquad \lfloor \beta \rfloor^\circ \mathsf{H} = \lfloor \beta \mathsf{H} \rfloor^\circ
$$

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

$$
\mathscr{C} \underset{\mathsf{R}}{\overset{\mathsf{L}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{D} \qquad \text{then} \qquad \mathscr{E}^{\mathscr{D}} \underset{-\mathsf{L}}{\overset{-\mathsf{R}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathscr{E}^{\mathscr{C}} \qquad\qquad \mathscr{E}^{\mathscr{D}}(\mathsf{FR}, \mathsf{G}) \cong \mathscr{E}^{\mathscr{C}}(\mathsf{F}, \mathsf{GL})
$$

Note that left and the right adjoint are swapped in the lifted adjunction. (This is because pre-composition $-\mathsf{F}$ is the arrow part of the *contravariant* functor $\mathscr{C}^{(-)} : \mathbf{Cat}^{\mathsf{op}} \to \mathbf{Cat}$.) The left adjunct, written $\lceil - \rceil^\circ$, and the right adjunct, written $\lceil - \rceil$, are defined

$$
\lceil \beta : \mathsf{FR} \to \mathsf{G} \rceil^\circ = \beta \mathsf{L} \cdot \mathsf{F}\eta \ , \qquad\qquad \lceil \alpha : \mathsf{F} \to \mathsf{GL} \rceil = \mathsf{G}\epsilon \cdot \alpha \mathsf{R} \ . \tag{8}
$$

As 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.

The fusion laws

$$
\begin{aligned}
\lceil \alpha \rceil^\circ \cdot \beta &= \lceil \alpha \cdot \beta \mathsf{R} \rceil^\circ && \lceil \alpha \rceil \cdot \beta \mathsf{R} = \lceil \alpha \cdot \beta \rceil && (9) \\
\alpha \mathsf{L} \cdot \lceil \beta \rceil^\circ &= \lceil \alpha \cdot \beta \rceil^\circ && \alpha \cdot \lceil \beta \rceil = \lceil \alpha \mathsf{L} \cdot \beta \rceil && (10)
\end{aligned}
$$

and the shift laws

$$
\mathsf{H} \lceil \beta \rceil^\circ = \lceil \mathsf{H}\beta \rceil^\circ \qquad\qquad \mathsf{H} \lceil \alpha \rceil = \lceil \mathsf{H}\alpha \rceil \tag{11}
$$

are dual to those for post-composition.

3.2 Transformation transformers

If we combine the adjuncts $\lfloor - \rfloor$ 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.

$$\lfloor \lceil \alpha \rceil \rfloor = \lceil \lfloor \alpha \rfloor \rceil \qquad (12) \qquad \lceil \lfloor \beta \rfloor^\circ \rceil^\circ = \lfloor \lceil \beta \rceil^\circ \rfloor^\circ$$

The straightforward proof makes use of the fact that $\mathsf{R}-$ and $-\mathsf{R}$ are functors.

$$\begin{array}{ll} & \lfloor \lceil \alpha \rceil \rfloor \\ = & \quad \{ \text{ definition of } \lceil - \rceil \ (8) \ \} \\ & \lfloor \mathsf{G}\epsilon \cdot \alpha\mathsf{R} \rfloor \\ = & \quad \{ \text{ definition of } \lfloor - \rfloor \ (3) \ \} \\ & \mathsf{R}(\mathsf{G}\epsilon \cdot \alpha\mathsf{R}) \cdot \eta\mathsf{FR} \\ = & \quad \{ \ \mathsf{R}- \text{ functor } (27) \ \} \\ & \mathsf{RG}\epsilon \cdot \mathsf{R}\alpha\mathsf{R} \cdot \eta\mathsf{FR} \\ = & \quad \{ \ -\mathsf{R} \text{ functor } (29) \ \} \\ & \mathsf{RG}\epsilon \cdot (\mathsf{R}\alpha \cdot \eta\mathsf{F})\mathsf{R} \\ = & \quad \{ \text{ definition of } \lceil - \rceil \ (8) \ \} \\ & \lceil \mathsf{R}\alpha \cdot \eta\mathsf{F} \rceil \\ = & \quad \{ \text{ definition of } \lfloor - \rfloor \ (3) \ \} \\ & \lceil \lfloor \alpha \rfloor \rceil \end{array}$$

As an aside, we also have $\lfloor \lceil \alpha \rceil^\circ \rfloor = \lceil \lfloor \alpha \rfloor \rceil^\circ$ and $\lceil \lfloor \beta \rfloor^\circ \rceil = \lfloor \lceil \beta \rceil \rfloor^\circ$. These are the adjuncts of the adjunction $\mathsf{L}-\mathsf{R} \dashv \mathsf{R}-\mathsf{L}$.

Let us now assume that L and R are endofunctors. Then we can nest $\lfloor - \rfloor$ and $\lceil - \rceil$ arbitrarily deep so that $\lceil \lfloor - \rfloor^m \rceil^n$ sends $\mathsf{L}^m\mathsf{F} \to \mathsf{GL}^n$ to $\mathsf{R}^n\mathsf{F} \to \mathsf{GR}^m$. We will refer to $\lceil \lfloor - \rfloor^m \rceil^n$ simply as a *transformer*. As 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. To get a better grip on iterated adjuncts it is useful to introduce generalisations of the units: $\epsilon_n : \mathsf{L}^n\mathsf{R}^n \to \mathsf{I}$ and $\eta_n : \mathsf{I} \to \mathsf{R}^n\mathsf{L}^n$ defined $\epsilon_0 = id_{\mathsf{I}} = \eta_0$ and

$$\epsilon \cdot \mathsf{L}\epsilon_n\mathsf{R} = \epsilon_{n+1} = \epsilon_n \cdot \mathsf{L}^n\epsilon\mathsf{R}^n \ , \qquad (13) \qquad \mathsf{R}^n\eta\mathsf{L}^n \cdot \eta_n = \eta_{n+1} = \mathsf{R}\eta_n\mathsf{L} \cdot \eta \ . \qquad (14)$$

The generalised counit ϵ_n builds a tower of ϵs decorated with Ls and Rs. For instance, ϵ_3 unfolds to $\epsilon \cdot \mathsf{L}\epsilon\mathsf{R} \cdot \mathsf{LL}\epsilon\mathsf{RR}$. The generalised units are the images of the identity on L^n and R^n, respectively.

$$\lceil id_{\mathsf{L}^n} \rceil^n = \epsilon_n = \lfloor id_{\mathsf{R}^n} \rfloor^{\circ n} \qquad (15) \qquad \lfloor id_{\mathsf{L}^n} \rfloor^n = \eta_n = \lceil id_{\mathsf{R}^n} \rceil^{\circ n} \qquad (16)$$

The proof of these laws proceeds by straightforward induction on n.

Using the generalised units we can characterise the n-fold adjuncts.

$$\lfloor \alpha : \mathsf{L}^n\mathsf{F} \to \mathsf{G} \rfloor^n = \mathsf{R}^n\alpha \cdot \eta_n\mathsf{F} \qquad (17) \qquad \lfloor \beta : \mathsf{F} \to \mathsf{R}^n\mathsf{G} \rfloor^{\circ n} = \epsilon_n\mathsf{G} \cdot \mathsf{L}^n\beta$$

$$\lceil \beta : \mathsf{FR}^n \to \mathsf{G} \rceil^{\circ n} = \beta\mathsf{L}^n \cdot \mathsf{F}\eta_n \qquad \lceil \alpha : \mathsf{F} \to \mathsf{GL}^n \rceil^n = \mathsf{G}\epsilon_n \cdot \alpha\mathsf{R}^n \qquad (18)$$

For the proofs we repeatedly appeal to fusion and shift. For the first identity we calculate

$$\begin{array}{ll} & \lfloor \alpha \rfloor^n \\ = & \quad \{ \ \lfloor - \rfloor \text{-fusion } (6), \ n \text{ times } \} \\ & \mathsf{R}^n\alpha \cdot \lfloor id_{\mathsf{L}^n\mathsf{F}} \rfloor^n \\ = & \quad \{ \ \lfloor - \rfloor \text{-shift } (7), \ n \text{ times } \} \\ & \mathsf{R}^n\alpha \cdot \lfloor id_{\mathsf{L}^n} \rfloor^n\mathsf{F} \\ = & \quad \{ \text{ characterisation of } \eta_n \ (16) \ \} \\ & \mathsf{R}^n\alpha \cdot \eta_n\mathsf{F} \ . \end{array}$$

The other calculations are similar.

Finally, the generalised units satisfy generalised triangle identities.

$$\epsilon_n \mathsf{L}^n \cdot \mathsf{L}^n \eta_n = id_{\mathsf{L}^n} \qquad (19) \qquad \mathsf{R}^n \epsilon_n \cdot \eta_n \mathsf{R}^n = id_{\mathsf{R}^n}$$

The proof proceeds by induction over n. **Case** 0**:** the identity simplifies to $id_{\mathsf{I}} \cdot id_{\mathsf{I}} = id_{\mathsf{I}}$. **Case** $n+1$**:** we reason

$$\begin{aligned}
& \epsilon_{n+1} \mathsf{L}^{n+1} \cdot \mathsf{L}^{n+1} \eta_{n+1} \\
= & \quad \{ \text{ definition of } \epsilon_{n+1} \text{ (13) and definition of } \eta_{n+1} \text{ (14) } \} \\
& (\epsilon \cdot \mathsf{L} \epsilon_n \mathsf{R}) \mathsf{L}^{n+1} \cdot \mathsf{L}^{n+1} (\mathsf{R}^n \eta \mathsf{L}^n \cdot \eta_n) \\
= & \quad \{ -\mathsf{L} \text{ functor (29) and } \mathsf{L}- \text{ functor (27) } \} \\
& \epsilon \mathsf{L}^{n+1} \cdot \mathsf{L} \epsilon_n \mathsf{R} \mathsf{L}^{n+1} \cdot \mathsf{L}^{n+1} \mathsf{R}^n \eta \mathsf{L}^n \cdot \mathsf{L}^{n+1} \eta_n \\
= & \quad \{ \text{ horizontal composition (35): } \epsilon_n : \mathsf{L}^n \mathsf{R}^n \to \mathsf{I} \text{ and } \eta : \mathsf{I} \to \mathsf{R}\mathsf{L} \} \\
& \epsilon \mathsf{L}^{n+1} \cdot \mathsf{L} \eta \mathsf{L}^n \cdot \mathsf{L} \epsilon_n \mathsf{L}^n \cdot \mathsf{L}^{n+1} \eta_n \\
= & \quad \{ -\mathsf{L} \text{ functor (29) and } \mathsf{L}- \text{ functor (27) } \} \\
& (\epsilon \mathsf{L} \cdot \mathsf{L} \eta) \mathsf{L}^n \cdot \mathsf{L} (\epsilon_n \mathsf{L}^n \cdot \mathsf{L}^n \eta_n) \\
= & \quad \{ \text{ triangle identity (1) and ex hypothesi } \} \\
& id_{\mathsf{L}^{n+1}} \ .
\end{aligned}$$

Thus prepared we can now turn to the heart of the matter.

4 Monads from comonads, comonads from monads

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, if it exists, is a comonad. The 'transformation transformers' of the previous section allow us to systematically turn the comonadic operations into monadic ones and vice versa.

$$r = \lfloor e \rfloor : \mathsf{I} \to \mathsf{R} \qquad (20) \qquad e = \lfloor r \rfloor^\circ : \mathsf{L} \to \mathsf{I}$$

$$j = \lfloor \lceil \lceil d \rceil \rceil \rfloor : \mathsf{R}\mathsf{R} \to \mathsf{R} \qquad (21) \qquad d = \lceil \lceil \lfloor j \rfloor^\circ \rceil^\circ \rceil^\circ : \mathsf{L} \to \mathsf{L}\mathsf{L}$$

Since the adjuncts are inverses, going round in a circle yields the original structure. Furthermore, comonadic programs of type $\mathsf{L}\, A \to B$ are in one-to-one correspondence to monadic programs of type $A \to \mathsf{R}\, B$. So in this particular situation, the choice between comonadic and monadic style is not a matter of expressiveness, it is purely a matter of personal taste. (Functional programmers seem to lean to the right.)

It remains to show that the comonadic laws imply the monadic laws and vice versa. For the proof it is sufficient to concentrate on natural transformations of type $\mathsf{L}^m \to \mathsf{L}^n$ and $\mathsf{R}^n \to \mathsf{R}^m$, respectively. We have seen in the previous section that these two types of transformations are in one-to-one correspondence, as well. In particular, the transformers send the identity on L^n to the identity on R^n and vice versa.

$$\lfloor \lceil id_{\mathsf{L}^n} \rceil^n \rfloor^n = id_{\mathsf{R}^n} \qquad (22) \qquad \lceil \lfloor id_{\mathsf{R}^n} \rfloor^{\circ n} \rceil^{\circ n} = id_{\mathsf{L}^n} \qquad (23)$$

This is a direct consequence of the characterisation of ϵ_n (15) and η_n (16). The transformers also preserve composition of natural transformations.

$$\lceil \lfloor \beta \cdot \alpha \rfloor^k \rceil^n = \lceil \lfloor \alpha \rfloor^k \rceil^m \cdot \lceil \lfloor \beta \rfloor^m \rceil^n \qquad (24) \qquad \lceil \lfloor \beta \cdot \alpha \rfloor^{\circ k} \rceil^{\circ n} = \lceil \lfloor \alpha \rfloor^{\circ k} \rceil^{\circ m} \cdot \lceil \lfloor \beta \rfloor^{\circ m} \rceil^{\circ n}$$

Note that the order of the natural transformations β and α is swapped on the right-hand sides. We will get back to this observation in a second. First, we reason

$$\begin{aligned}
& \lfloor \lceil \alpha \rceil^m \rfloor^k \cdot \lceil \lfloor \beta \rfloor^m \rceil^n \\
= & \quad \{ \ \lfloor - \rfloor \text{-fusion (5), } k \text{ times } \} \\
& \lfloor \ \lceil \alpha \rceil^m \cdot \mathsf{L}^k \lceil \lfloor \beta \rfloor^m \rceil^n \ \rfloor^k
\end{aligned}$$

$$
\begin{array}{ll}
= & \{\ \lceil - \rceil\text{-shift (7), } n \text{ times }\} \\
& \lfloor\ \lceil \alpha \rceil^m \cdot \lceil \mathsf{L}^k \lfloor \beta \rfloor^m \rceil^n\ \rfloor^k \\
= & \{\ \lceil - \rceil\text{-fusion (10), } n \text{ times }\} \\
& \lfloor \lceil\ \lceil \alpha \rceil^m \mathsf{L}^n \cdot \mathsf{L}^k \lfloor \beta \rfloor^m\ \rceil^n \rfloor^k \\
= & \{\ \text{claim, see below }\} \\
& \lfloor \lceil\ \beta \cdot \alpha\ \rceil^n \rfloor^k\ .
\end{array}
$$

The claim can be shown as follows.

$$
\begin{array}{ll}
& \lceil \alpha \rceil^m \mathsf{L}^n \cdot \mathsf{L}^k \lfloor \beta \rfloor^m \\
= & \{\ \text{characterisation of } \lceil - \rceil^m \text{ (18) and characterisation of } \lfloor - \rfloor^m \text{ (17) }\} \\
& (\epsilon_m \cdot \alpha \mathsf{R}^m) \mathsf{L}^n \cdot \mathsf{L}^k (\mathsf{R}^m \beta \cdot \eta_m) \\
= & \{\ -\mathsf{L} \text{ functor (29) and } \mathsf{L}- \text{ functor (27) }\} \\
& \epsilon_m \mathsf{L}^n \cdot \alpha \mathsf{R}^m \mathsf{L}^n \cdot \mathsf{L}^k \mathsf{R}^m \beta \cdot \mathsf{L}^k \eta_m \\
= & \{\ \text{horizontal composition (35): } \alpha : \mathsf{L}^k \to \mathsf{L}^m \text{ and } \beta : \mathsf{L}^m \to \mathsf{L}^n\ \} \\
& \epsilon_m \mathsf{L}^n \cdot \mathsf{L}^m \mathsf{R}^m \beta \cdot \alpha \mathsf{R}^m \mathsf{L}^m \cdot \mathsf{L}^k \eta_m \\
= & \{\ \text{horizontal composition (35), twice:} \\
& \quad \epsilon_m : \mathsf{L}^m \mathsf{R}^m \to \mathsf{I} \text{ and } \beta : \mathsf{L}^m \to \mathsf{L}^n, \text{ and } \alpha : \mathsf{L}^k \to \mathsf{L}^m \text{ and } \eta_m : \mathsf{R}^m \mathsf{L}^m \to \mathsf{I}\ \} \\
& \beta \cdot \epsilon_m \mathsf{L}^m \cdot \mathsf{L}^m \eta_m \cdot \alpha \\
= & \{\ \text{generalised triangle identity (19) }\} \\
& \beta \cdot \alpha
\end{array}
$$

One way to look at these properties is to view the transformers as the arrow parts of two *contravariant* functors—contravariant because an adjunction trades L in the *source* for R in the *target* of an arrow. Specifically, consider the full subcategory $\mathscr{L}$ of $\mathscr{C}^{\mathscr{C}}$ whose objects are the composites L^n and whose arrows are the natural transformations between them. The category $\mathscr{R}$ whose objects are the composites R^n is defined likewise. Then the contravariant functors $[\![-]\!] : \mathscr{L} \to \mathscr{R}^{\mathsf{op}}$ and $[\![-]\!]^\circ : \mathscr{R}^{\mathsf{op}} \to \mathscr{L}$ defined

$$
\begin{array}{cc}
[\![\mathsf{L}^n]\!] = \mathsf{R}^n & [\![\mathsf{R}^n]\!]^\circ = \mathsf{L}^n \\
[\![\alpha : \mathsf{L}^m \to \mathsf{L}^n]\!] = \lceil \lfloor \alpha \rfloor^m \rceil^n : \mathsf{R}^n \to \mathsf{R}^m & [\![\alpha : \mathsf{R}^n \to \mathsf{R}^m]\!]^\circ = \lceil \lfloor \alpha \rfloor^{\circ n} \rceil^{\circ m} : \mathsf{L}^m \to \mathsf{L}^n
\end{array}
$$

are isomorphisms of categories. Thus it is little surprise that the comonadic structure is transmogrified into a monadic structure and vice versa. To actually prove this we need one final ingredient, the flip laws.

$$
\lfloor \lceil \mathsf{L}\alpha \rceil^{n+1} \rfloor^{m+1} = \lfloor \lceil \alpha \rceil^n \rfloor^m \mathsf{R} \qquad (25) \qquad \lfloor \lceil \alpha \mathsf{L} \rceil^{n+1} \rfloor^{m+1} = \lfloor \mathsf{R} \lceil \alpha \rceil^n \rfloor^m
$$

Post-composition with L is mapped to pre-composition with R, and dually, pre-composition with L is mapped to post-composition with R.

$$
\begin{array}{ll}
& \lfloor \lceil \mathsf{L}\alpha \rceil^{n+1} \rfloor^{m+1} \\
= & \{\ \text{reorganise brackets (12) }\} \\
& \lfloor\ \lceil \lfloor \lceil \mathsf{L}\alpha \rceil^n \rfloor \rceil\ \rfloor^m \\
= & \{\ \lceil - \rceil\text{-shift (11), } n \text{ times }\} \\
& \lfloor\ \lceil \lfloor \mathsf{L} \lceil \alpha \rceil^n \rfloor \rceil\ \rfloor^m \\
= & \{\ \lfloor - \rfloor\text{-fusion (5) }\}
\end{array}
$$

$$
\begin{aligned}
& \lfloor\ \lceil \lfloor id_{\mathsf{L}} \rfloor \cdot \lceil \alpha \rceil^n \rceil\ \rfloor^m \\
= & \quad \{\ \lceil - \rceil\text{-fusion (10)}\ \} \\
& \lfloor\ \lceil \lfloor id_{\mathsf{L}} \rfloor \rceil \cdot \lceil \alpha \rceil^n \mathsf{R}\ \rfloor^m \\
= & \quad \{\ \text{preservation of identity (23)}\ \} \\
& \lfloor\ \lceil \alpha \rceil^n \mathsf{R}\ \rfloor^m \\
= & \quad \{\ \lfloor - \rfloor\text{-shift (7), } m \text{ times}\ \} \\
& \lfloor \lceil \alpha \rceil^n \rfloor^m \mathsf{R}
\end{aligned}
$$

It is time to pick the fruit. The proof that the first comonadic unit law is equivalent to the first monadic unit law is now a breeze.

$$
\begin{aligned}
& \mathsf{L}e \cdot d = id_{\mathsf{L}} \\
\iff & \quad \{\ \text{inverses}\ \} \\
& \lfloor \lceil \mathsf{L}e \cdot d \rceil \rfloor = \lfloor \lceil id_{\mathsf{L}} \rceil \rfloor \\
\iff & \quad \{\ \text{preservation of composition (24) and preservation of identity (22)}\ \} \\
& \lfloor \lceil \lceil d \rceil \rceil \rfloor \cdot \lfloor \lfloor \lceil \mathsf{L}e \rceil \rfloor \rfloor = id_{\mathsf{R}} \\
\iff & \quad \{\ \text{flip law (25)}\ \} \\
& \lfloor \lceil \lceil d \rceil \rceil \rfloor \cdot \lfloor e \rfloor \mathsf{R} = id_{\mathsf{R}} \\
\iff & \quad \{\ \text{definition of } j \text{ and definition of } r\ \} \\
& j \cdot r\mathsf{R} = id_{\mathsf{R}}
\end{aligned}
$$

The other two proofs consist of exactly the same steps.

Now, our running example, the curry adjunction, provides an example, where the left adjoint $\mathsf{L} = - \times X$ is also a comonad. Its operations are defined

$$
\begin{aligned}
e &= \mathit{outl}\ , \\
d &= \mathit{id} \vartriangle \mathit{outr}\ .
\end{aligned}
$$

The so-called *product comonad* provides contextual information, e discards this information and d duplicates it. Some straightforward calculations show that d and e thus defined are indeed natural transformations and that they satisfy the three comonad laws. For the curry adjunction the transformers simplify to $\lfloor \alpha \rfloor\, A = \Lambda\,(\alpha\, A)$ and $\lceil \alpha : \mathsf{F} \to \mathsf{GL} \rceil\, B = \mathsf{G}\, \mathit{app} \cdot \alpha\,(\mathsf{R}\, B)$. The central result then implies that L's right adjoint $\mathsf{R} = (-)^X$ is a monad with operations

$$
\begin{aligned}
r &= \lfloor \mathit{outl} \rfloor = \Lambda\, \mathit{outl}\ , \\
d &= \lfloor \lceil \lceil \mathit{id} \vartriangle \mathit{outr} \rceil \rceil \rfloor = \Lambda\,(\mathit{app} \cdot (\mathit{app} \times X) \cdot (\mathit{id} \vartriangle \mathit{outr})) = \Lambda\,(\mathit{app} \cdot (\mathit{app} \vartriangle \mathit{outr}))\ .
\end{aligned}
$$

The resulting structure is known as the *reader monad*. The theory confirms our intuition that the product comonad and the reader monad solve the same problem. To reiterate, programs of type $A \times X \to B$ that are structured using the product comonad are in one-to-one correspondence to programs of type $A \to B^X$ that build on the reader monad.

Every comonad and every monad comes equipped with additional operations—these are related too. For instance, the product comonad might provide a getter and an update operation:

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

where Δ_X is the constant functor that maps an arbitrary object to X. The transforms of *get* and *update* correspond to operations called *ask* and *local* in the Haskell monad transformer library [1].

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

The properties of the operations transfer as well. As an example, *update* satisfies functor-like properties.

$$update\ id = id\ , \qquad update\ (f \cdot g) = update\ g \cdot update\ f\ .$$

This is because $id \times f$ is actually the arrow part of a functor, namely $A \times -$. It is only that *update* has a more restricted type because, for simplicity, we choose not to parametrise the product comonad with the type X of states. The corresponding properties of the reader monad are

$$local\ id = id\ , \qquad local\ (f \cdot g) = local\ g \cdot local\ f\ .$$

Note that g and f are swapped on the right-hand side: $\Lambda\,(app \cdot (id \times f))$ is the arrow part of a contravariant functor, namely $B^{(-)}$. The framework of adjunctions explains why the *covariant* functor $A \times -$ is mapped to the *contravariant* functor $B^{(-)}$. Using the concept of an adjunction with a parameter this can be made precise. We resist the temptation to do so because it is time to wrap up. Before we do this, here is a final twist.

5 The wrong way round

Does the translation also work if the left adjoint is simultaneously a *monad*? Yes and no. The transformers happily take the monadic operations to comonadic ones. So L's right adjoint is indeed a comonad. However, monadic programs of type $A \to \mathsf{L}\,B$ are certainly not in one-to-one correspondence to comonadic programs of type $\mathsf{R}\,A \to B$.

Let us explore the implications, working through a concrete example. If X is a monoid with operations $[\,] : X$ and $(+\!\!+) : X \times X \to X$, then $\mathsf{L} = - \times X$, our product comonad, also has the structure of a monad.[1]

$$\begin{aligned} r\ a &= (a, [\,])\ , \\ j\ ((a, x_1), x_2) &= (a, x_1 +\!\!+ x_2)\ . \end{aligned}$$

(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 then a comonad, lovingly called the "read from a monoid" comonad. For the operations, we unfold $\lceil r \rceil = app \cdot r$ and $\lfloor \lfloor \lceil j \rceil \rfloor \rfloor = \Lambda\,(\Lambda\,(app \cdot j))$.

$$\begin{aligned} e\ f &= f\ [\,]\ , \\ d\ f &= \lambda\, x_1\ .\ \lambda\, x_2\ .\ f\ (x_1 +\!\!+ x_2)\ . \end{aligned}$$

This worked out nicely. However, we cannot translate the accompanying infrastructure of the writer monad. Consider, for instance, the *write* operation.

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

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

6 Conclusion

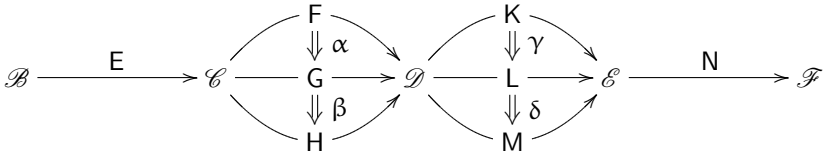
Shall I structure my programs using comonads or using monads? We have seen that sometimes this is a matter of personal taste. In general, the answer is to use both structures and both at the same time. Of course, one has to ensure that effectful and context-dependent computations interact nicely. This can be accomplished using a so-called distributive law of a comonad over a monad. As an example, clocked data-flow computations can be described in such a setting [2].

[1] I am grateful to Jeremy Gibbons for suggesting this example.

A Composition of functors and natural transformations

This appendix contains supplementary material. It is intended primarily as a reference, so that the reader can re-familiarise themselves with the category theory that is utilised in this pearl.

Specifically, we introduce composition of functors and natural transformations. We shall use the following entities to frame the discussion ($\mathsf{F}, \mathsf{G} : \mathscr{C} \to \mathscr{D}$ are parallel functors, $\alpha : \mathsf{F} \to \mathsf{G}$ is a natural transformation between them etc).



Functors can be composed, written simply using juxtaposition KF. Rather intriguingly, the operation $\mathsf{K}-$, post-composing a functor K, is itself functorial: the higher-order functor $\mathsf{K}- : \mathscr{D}^{\mathscr{C}} \to \mathscr{E}^{\mathscr{C}}$ maps the functor F to the functor KF and the natural transformation α to the natural transformation $\mathsf{K}\alpha$ defined $(\mathsf{K}\alpha)\, A = \mathsf{K}\,(\alpha\, A)$. Post-composition dualizes to pre-composition: the higher-order functor $-\mathsf{E} : \mathscr{D}^{\mathscr{C}} \to \mathscr{D}^{\mathscr{B}}$ maps the functor F to the functor FE and the natural transformation α to the natural transformation $\alpha\mathsf{E}$ defined $(\alpha\mathsf{E})\, A = \alpha\,(\mathsf{E}\, A)$. (The reader should convince themselves that $\mathsf{K}\alpha : \mathsf{KF} \to \mathsf{KG}$ and $\alpha\mathsf{E} : \mathsf{FE} \to \mathsf{GE}$ are again natural transformations.) Here are the functor laws spelled out.

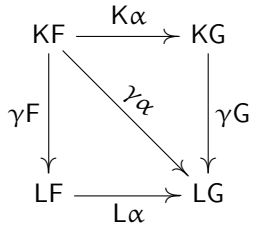
$$\mathsf{K}id_{\mathsf{F}} = id_{\mathsf{KF}} \tag{26}$$
$$\mathsf{K}(\beta \cdot \alpha) = (\mathsf{K}\beta) \cdot (\mathsf{K}\alpha) \tag{27}$$
$$id_{\mathsf{F}}\mathsf{E} = id_{\mathsf{FE}} \tag{28}$$
$$(\beta \cdot \alpha)\mathsf{E} = (\beta\mathsf{E}) \cdot (\alpha\mathsf{E}) \tag{29}$$

Altogether, we have three different forms of composition: KF, $\gamma\mathsf{F}$ and $\mathsf{K}\alpha$. They are 'pseudo-associative' and have the functor Id as their neutral element.

$$\gamma(\mathsf{FE}) = (\gamma\mathsf{F})\mathsf{E} \tag{30}$$
$$\mathsf{K}(\beta\mathsf{E}) = (\mathsf{K}\beta)\mathsf{E} \tag{31}$$
$$\mathsf{N}(\mathsf{M}\alpha) = (\mathsf{NM})\alpha \tag{32}$$
$$\mathsf{Id}\alpha = \alpha \tag{33}$$
$$\alpha\mathsf{Id} = \alpha \tag{34}$$

This means that we can freely drop parentheses when composing compositions.

Given two natural transformations $\alpha : \mathsf{F} \to \mathsf{G}$ and $\gamma : \mathsf{K} \to \mathsf{L}$, there are two ways to turn a KF into a LG structure.



The diagram commutes since γ is natural:

$$\begin{aligned}
& ((\gamma\mathsf{G}) \cdot (\mathsf{K}\alpha))\, X \\
= \quad & \{ \text{ definition of compositions } \} \\
& \gamma\,(\mathsf{G}\, X) \cdot \mathsf{K}\,(\alpha\, X) \\
= \quad & \{ \ \gamma \text{ is natural: } \mathsf{L}\, h \cdot \gamma\, A = \gamma\, B \cdot \mathsf{K}\, h \ \} \\
& \mathsf{L}\,(\alpha\, X) \cdot \gamma\,(\mathsf{F}\, X) \\
= \quad & \{ \text{ definition of compositions } \} \\
& ((\mathsf{L}\alpha) \cdot (\gamma\mathsf{F}))\, X \quad .
\end{aligned}$$

The diagonal is called the *horizontal composition* of natural transformations, denoted $\gamma\alpha$.

$$(\gamma \mathsf{G}) \cdot (\mathsf{K}\alpha) = \gamma\alpha = (\mathsf{L}\alpha) \cdot (\gamma \mathsf{F}) \tag{35}$$

As an aside, the definition witnesses the fact that functor composition $\mathscr{E}^{\mathscr{D}} \times \mathscr{D}^{\mathscr{C}} \to \mathscr{E}^{\mathscr{C}}$ is a bi-functor: (35) defines its action on arrows.

References

1. Gill, A.: Monad transformer library (mtl package) (2010) `http://hackage.haskell.org/package/mtl`.
2. Uustalu, T., Vene, V.: The essence of dataflow programming. In Horváth, Z., ed.: Central European Functional Programming School. Volume 4164 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2006) 135–167