Electronic Communications of the EASST
Volume 49 (2012)

Proceedings of the
First International Workshop on
Bidirectional Transformations
(BX 2012)

Relating Algebraic and Coalgebraic Descriptions of Lenses

Jeremy Gibbons and Michael Johnson

16 pages

Guest Editors: Frank Hermann, Janis Voigtländer
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Relating Algebraic and Coalgebraic Descriptions of Lenses

Jeremy Gibbons[1] **and Michael Johnson**[2]

[1] http://www.cs.ox.ac.uk/jeremy.gibbons/
Department of Computer Science
University of Oxford, UK

[2] http://www.comp.mq.edu.au/~mike/
Department of Computing
Macquarie University, AU

School of Mathematics and Statistics
University of Sydney, AU

Abstract: Lenses are a heavily studied form of bidirectional transformation, with diverse applications including database view updating, software development and memory management. Previous work has explored lenses category-theoretically, and established that the category of lenses for a fixed 'view' V is, up to isomorphism, the category of algebras for a particular monad on $\mathbf{set}/V$. It has recently been shown that lenses are the coalgebras for the comonad generated by the cartesian closure adjunction on **set**. In this paper, we present an equational proof of the coalgebra correspondence, note that the algebra correspondence extends to arbitrary categories with products and that the coalgebra correspondence extends to arbitrary cartesian closed categories, and show that both correspondences extend to isomorphisms of categories. The resulting isomorphism between a category of algebras and a category of coalgebras is unexpected, and we analyze its underlying generality and the particularity that restricts its applicability. We end with remarks about the utility of the two different treatments of lenses, especially for obtaining further, more realistic, generalizations of the notion of lens.

Keywords: Lens, view update, algebra, coalgebra, monad, comonad, slice category

1 Introduction

The *view update problem* [BS81] is a fundamental problem in database design. Given a complex database schema specifying multiple tables, with integrity constraints on foreign key references between them, one usually wants to carry out specific data-dependent activities on a simpler and transient 'view' of the database, perhaps determined by a query over the persistent 'source' database. But if one then wishes to perform an update to the data, expressed solely in terms of the view, then somehow this needs to be translated into a 'corresponding' update on the source.

The problem is widely studied, not only in the database world. It also underlies 'model–view–controller'-style applications [KP88], which must maintain consistency between a graphical user interface and an underlying network of data objects, and the related issue of managing transformations between *ad hoc* data formats [FGM+07, FMW10]. In a more symmetric presentation,

it describes the synchronization of models from multiple perspectives of the same system design [Ste10]. For a recent summary of the state of the art, see [CFH+09].

From an engineering perspective, one would rather not write separate programs for the two directions of transformation—generating the view from the source, and updating the source from a modified view. These two programs would be closely coupled, and the duplication in effort would be inefficient and prone to inconsistency. Rather, one would like to program in a *bidirectional* style, writing one specification that serves to describe both transformations simultaneously.

One prominent model of such an (asymmetric) bidirectional programming style is the work on 'lenses' by Pierce *et al.* [FGM+07]. For a source type S and view type V, a lens consists of a pair of related total functions: a 'get' function $g : S \to V$, and a 'put' function $p : S \times V \to S$. Note the asymmetry, in that the 'put' function takes not only an updated view, but also the original source—it is usually the case that the view omits some of the data from the source (that is the whole point of the endeavour, after all), and so it is unreasonable to ask for a useful backwards function of type $V \to S$; in contrast, all the data needed for the view is recorded in the source, so there is no problem with a forwards function of type $S \to V$. (Thus, lenses are usually intended for asymmetric contexts, in which one data format is a 'master' copy and the other a 'slave'; however, there are some recent developments on symmetrizing the theory [HPW11].)

The 'get' and 'put' functions should, of course, be related to each other, enjoying a kind of inverse property. In a *well-behaved lens*, the two functions satisfy the 'get–put' and 'put–get' laws:

$$\begin{array}{lll} p(s, g\,s) & = s & \text{-- get–put} \\ g(p(s,v)) & = v & \text{-- put–get} \end{array}$$

Informally, the get–put law states that if one gets a view from the source and puts it straight back again without modification, the source remains unchanged; the put–get law states that if one puts an arbitrary view v into the source, one can get it back again without loss. These two laws are rather weak, in that they say nothing about what happens under modifications; in addition, the lens is *very well-behaved* if it also satisfies the 'put–put' law:

$$p(p(s,u),v) = p(s,v) \quad \text{-- put–put}$$

which states that putting back a second view v after a first view u completely overwrites any consequences of putting back u. (Actually, the put–put law as just stated turns out to be too strong for many applications: in classical circumstances it implies that the source S can be partitioned into two completely separate factors, the view V and a complement C, which can each be independently modified. We will return to this issue towards the end of the paper.)

Pierce *et al.* [FGM+07] present a combinator-style language for defining lenses, based on a handful of primitive lens constructs (such as constants and projections) and some general-purpose operations (such as sequential and parallel composition) for composing more complicated lenses out of simpler ones. An alternative, more semantic, approach, exemplified by Voigtländer [Voi09], is to allow arbitrary programming constructs in defining the 'get' function, and to infer the appropriate 'put' function (if indeed one exists) by means of parametricity. Either way, the design of the framework guarantees that the resulting lenses are very well-behaved, and it is the structure of these very well-behaved lenses that is our subject.

Recent work by Johnson *et al.* [JRW10] has shown that the very well-behaved lenses are precisely the algebras for a particular monad. Because the relevant monad can be defined on any category with products, this approach provides an immediate generalisation of lenses (parametric on the base category), and it unifies several formerly distinct lens-like notions. Also recently, O'Connor has observed in a blog posting [O'C10] that the same very well-behaved lenses are precisely the coalgebras for a particular comonad. The main goal of this paper is to understand O'Connor's observation in the light of the work of Johnson *et al.* In particular we seek to establish the breadth of applicability of the coalgebraic description of lenses—to determine what structure on a base category is required to state and prove the correspondence between lens laws and coalgebras—and to explore the relationship between the algebraic and the coalgebraic approaches.

In the next section we review the algebraic approach to lenses. Then, in Section 3 we study the coalgebraic approach in detail—although O'Connor's note includes a Coq proof script demonstrating the equivalence in the category **set**, we provide here a category theoretic proof of the correspondence between lens laws and coalgebras, and by developing that proof using nothing more than the properties of a cartesian closed category demonstrate that the result is parametric on the base category for arbitrary cartesian closed categories. Section 4 presents a detailed exploration of the rather unexpected correspondence between the algebraic and coalgebraic models of lenses. Finally, Section 5 draws out the implications from this correspondence and the earlier sections for the further development of lenses and their generalizations.

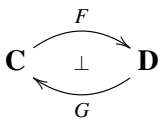
2 Lenses as algebras

In this section, we briefly summarize the presentation of very well-behaved lenses as $\Delta\Sigma$-algebras [JRW10].

2.1 Adjunctions, products, monads, algebras, slices

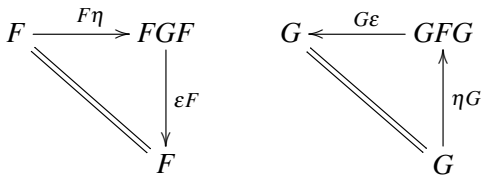
First, we review the basic categorical concepts that we will need: adjunctions, products, monads, algebras, and slices. We assume that the reader is already moderately familiar with these notions—the following nutshell summary is likely to be too dense for a first introduction—but include this reminder not least in order to fix notation. For more detail, see any standard textbook on category theory, such as [ML71] or [Pie91].

Given two categories $\mathbf{C}$ and $\mathbf{D}$, functors $F : \mathbf{C} \to \mathbf{D}$ and $G : \mathbf{D} \to \mathbf{C}$ form an *adjunction* when there is an isomorphism of homsets $\mathbf{D}(FA, B) \simeq \mathbf{C}(A, GB)$ that is natural in A and B. We write $F \dashv G$, and say that 'F is left adjoint to G'.



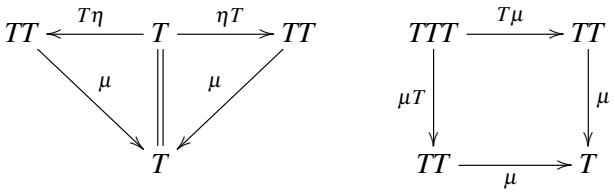
The witness to the isomorphism is a bijective *transposition* $\phi_{F \dashv G}$, taking an arrow in $\mathbf{D}(FA, B)$ to its image in $\mathbf{C}(A, GB)$. In particular, the 'unit' $\eta : 1_{\mathbf{C}} \dot{\to} GF$ and 'counit' $\varepsilon : FG \dot{\to} 1_{\mathbf{D}}$ of the adjunction are obtained from the transpositions $\eta_A = \phi_{F \dashv G}(1_{FA}) : A \to GFA$ and $\varepsilon_B = \phi^{-1}_{F \dashv G}(1_{GB})$:

$FGB \to B$ of identity arrows 1_{FA} in $\mathbf{D}$ and 1_{GB} in $\mathbf{C}$, and satisfy the following two triangle identities:



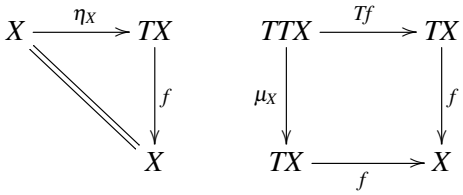
Throughout the paper, we will assume a *cartesian* category $\mathbf{C}$, that is, a fixed terminal object 1 and binary product $A \times B$ for each pair of objects A, B. We will write π_0 and π_1 for the projections from the product, and $\langle f, g \rangle$ for the tupling morphism with components f and g. (As is well known, products arise as the right adjoint of the diagonal functor $\mathbf{C} \to \mathbf{C} \times \mathbf{C}$; then $\langle f, g \rangle$ is the right adjunct of the arrow (f, g) in $\mathbf{C} \times \mathbf{C}$. The terminal object arises similarly, from the right adjoint of the functor $\mathbf{C} \to \mathbf{1}$.)

A *monad* $\mathsf{T} = (T, \eta, \mu)$ on a category $\mathbf{C}$ consists of a functor $T : \mathbf{C} \to \mathbf{C}$ with 'unit' and 'multiplication' natural transformations $\eta : 1_{\mathbf{C}} \dot{\to} T$ and $\mu : TT \dot{\to} T$ that satisfy the following unit and associativity identities:

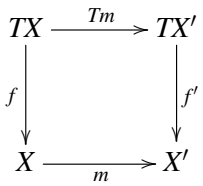


Every adjunction $F \dashv G$ determines a monad (GF, η, μ), in which the unit η of the monad is the unit of the adjunction, and the multiplication μ of the monad is defined in terms of the counit of the adjunction by $\mu = G\varepsilon F$.

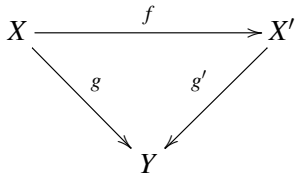
An *algebra* (X, f) for a monad (T, η, μ) on category $\mathbf{C}$ is an object X of $\mathbf{C}$ with an arrow, often called an 'action', $f : TX \to X$ satisfying the following two identities (also called unit and associativity):



A *morphism of algebras* from a T-algebra (X, f) to a T-algebra (X', f') is an arrow $m : X \to X'$ making the following diagram commute:



For a fixed object Y of $\mathbf{C}$, the *slice category* $\mathbf{C}/Y$ has as objects the arrows $g : X \to Y$ of $\mathbf{C}$, and as arrows from g to $g' : X' \to Y$ the arrows $f : X \to X'$ of $\mathbf{C}$ such that $g'f = g$—so arrows in $\mathbf{C}/Y$ are commuting triangles to vertex Y in $\mathbf{C}$:



2.2 View operations induce an adjunction

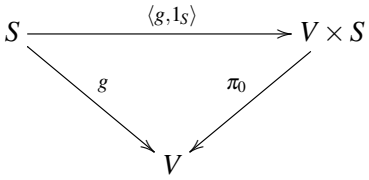
The view update problem is expressed in terms of mappings from a source S to a view V, so it is natural to explore models of lenses based on the slice category $\mathbf{C}/V$. The arrows $f : g \to g'$ in $\mathbf{C}/V$ make the triangle $g'f = g$ commute; one might think of them as updates on the source that leave the view unchanged. There is an obvious forgetful functor $\Sigma_V : \mathbf{C}/V \to \mathbf{C}$ that discards the slice structure, defined by $\Sigma_V g = S$ (when $g : S \to V$) and $\Sigma_V f = f$—named in this way because in effect it sums up all the slices. (We will usually drop the subscript V.)

To obtain a functor in the opposite direction, we assume that $\mathbf{C}$ has finite products, and define $\Delta_V : \mathbf{C} \to \mathbf{C}/V$ by $\Delta_V S = \pi_0 : V \times S \to V$ and $\Delta_V f = 1_V \times f$. (Again, we will usually drop the subscript V.) Note that $\Delta\Sigma g = \pi_0 : V \times S \to V$ when $g : S \to V$, discarding the original view function g on S and replacing it with the trivial view π_0 on $V \times S$.

The two functors Σ and Δ form an adjunction $\Sigma \dashv \Delta$. For $g : S \to V$ in $\mathbf{C}$, the isomorphism between homsets

$$\mathbf{C}(\Sigma g, C) \simeq (\mathbf{C}/V)(g, \Delta C)$$

amounts to the bijection between arrows $S \to C$ and arrows $f : S \to V \times C$ of $\mathbf{C}$ for which $\pi_0 f = g$. The g'th component $\eta_g : g \to \Delta\Sigma g$ of the unit of the adjunction is the arrow $\langle g, 1_S \rangle : S \to V \times S$ in $\mathbf{C}$, which makes the triangle

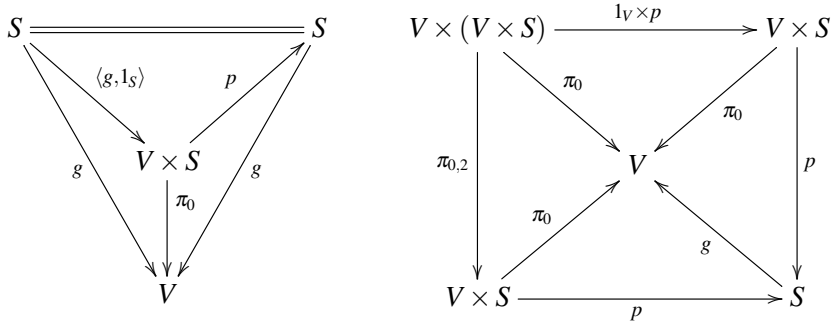


commute, as required.

The adjunction $\Sigma \dashv \Delta$ determines a monad $\mathsf{T}_{\Delta\Sigma} = (\Delta\Sigma, \eta, \mu)$ on $\mathbf{C}/V$. The unit of the monad is the unit η of the adjunction; the g'th component $\mu_g : \Delta\Sigma\Delta\Sigma g \to \Delta\Sigma g$ of the multiplication is the arrow $\langle \pi_0, \pi_1 \pi_1 \rangle : V \times (V \times S) \to V \times S$ in $\mathbf{C}$. (We will abbreviate $\langle \pi_0, \pi_1 \pi_1 \rangle$ to '$\pi_{0,2}$' in the sequel.)

2.3 Lenses are the algebras for the $\Delta\Sigma$ monad

A $\Delta\Sigma$-algebra is a pair (g, p) with g an object and p an arrow $\Delta\Sigma g \to g$, such that $p\eta.g = 1_g$ and $p\mu.g = p(\Delta\Sigma p)$, all read in $\mathbf{C}/V$. The result of unpacking these ingredients in $\mathbf{C}$ is shown in the tetrahedron and pyramid below, recalling that $\eta_g = \langle g, 1_S \rangle$, $\mu = \pi_{0,2}$, and $\Delta\Sigma p = 1_V \times p$.



The lower right face of the tetrahedron and the lower and right faces of the pyramid all capture the fact that p is an arrow in the slice category; the upper face of the tetrahedron is the unit law of the algebra; and the base of the pyramid is the associativity law. The remaining faces commute trivially, by projections.

Now, the 'get' and 'put' operations of a very well-behaved lens are (up to isomorphism) precisely the data required for a $\Delta\Sigma$-algebra. We specialize the category $\mathbf{C}$ to **set**, because the usual definition of lenses is as total functions between sets. Also, to make the relationship more obvious, we swap the two arguments of the 'put' function as presented in Section 1; so we have $g : S \to V$ and $p : V \times S \to S$. Then the 'get–put' law

$$p(g\,s, s) = s$$

when expressed in a pointfree style becomes $p\langle g, 1\rangle = 1$, which is precisely the unit law of the algebra. The 'put–get' law

$$g(p(v, s)) = v$$

becomes $gp = \pi_0$, which is the statement that $p : \Delta\Sigma g \to g$ in $\mathbf{set}/C$. Finally, the 'put–put' law

$$p(v, p(u, s)) = p(v, s)$$

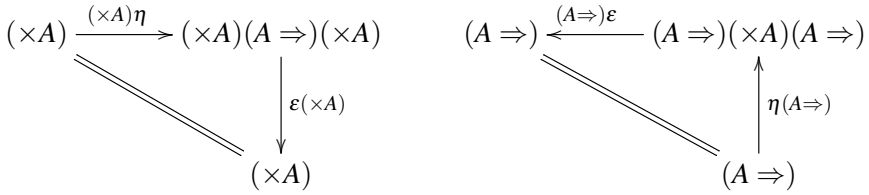
becomes $p(1 \times p) = p\pi_{0,2}$, which is the associativity property of the algebra.

3 Lenses as coalgebras

In this section, we provide a category-theoretic proof of the correspondence between very well-behaved lenses and coalgebras, showing that the result is parametric on the base cartesian closed category.

3.1 Cartesian closure, comonads, coalgebras

Within Section 3 of this paper, we make the additional assumption of *cartesian closure*: for any object A, there is an adjunction $(\times A) \dashv (A \Rightarrow)$ between an 'exponent' functor $(A \Rightarrow)$ and the product $(\times A)$. The unit $\eta : 1 \dot{\to} (A \Rightarrow)(\times A)$ and counit $\varepsilon : (\times A)(A \Rightarrow) \dot{\to} 1$ make the following two diagrams commute:



The B'th components of the natural transformations are

$$\eta_B : B \to A \Rightarrow (B \times A)$$
$$\varepsilon_B : (A \Rightarrow B) \times A \to B$$

so, in functional programming terms, the unit η is a kind of curried pairing operator, and the counit ε is function application. (Expressed pointwise, as arrows in **set**, the triangle equalities become:

$$\varepsilon_{B\times A}(\eta_B(b), a) = (b, a)$$
$$\varepsilon_B(\eta_{A\Rightarrow B}(f))(a) = f(a)$$

and the actions of the two adjoint functors on arrows reduce pointwise to $(f \times A)(b,a) = (f(b),a)$ and $(A \Rightarrow f)(g) = f \cdot g$. But throughout Section 3, we will stick to the high ground of a cartesian closed category in general, using **set** only for illustrations.)

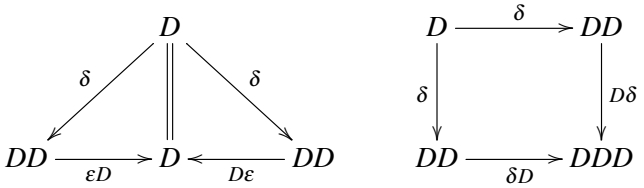
We will make use of curried versions of the pair projections, defined by

$$\mathsf{const}_0 = (A \Rightarrow \pi_0)\eta_B : B \to (A \Rightarrow B)$$
$$\mathsf{const}_1 = (A \Rightarrow \pi_1)\eta_B : B \to (A \Rightarrow A)$$

and note that const_1 is a left zero of composition—for $f : C \to B$,

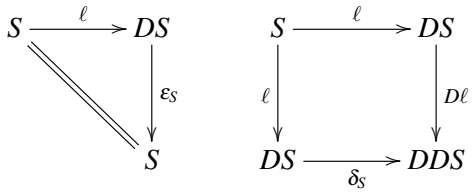
$$\begin{array}{ll} & \mathsf{const}_1 f \\ = & \quad [\![\text{ definition of } \mathsf{const}_1 \ ]\!] \\ & (A \Rightarrow \pi_1)\eta_B f \\ = & \quad [\![\text{ naturality of } \eta \ ]\!] \\ & (A \Rightarrow \pi_1)(A \Rightarrow (f \times A))\eta_C \\ = & \quad [\![\text{ functors; naturality of } \pi_1 \ ]\!] \\ & (A \Rightarrow \pi_1)\eta_C \\ = & \quad [\![\text{ definition of } \mathsf{const}_1 \ ]\!] \\ & \mathsf{const}_1 \end{array}$$

A *comonad* (D, ε, δ) on a category $\mathbf{C}$ consists of a functor $D : \mathbf{C} \to \mathbf{C}$ with two natural transformations $\varepsilon : 1 \dot{\to} D$ and $\delta : DD \dot{\to} D$ that satisfy the following counit and coassociativity identities:

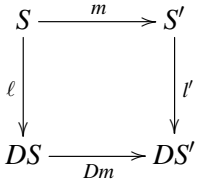


Every adjunction $F \dashv G$ determines a comonad $(FG, \varepsilon, \delta)$, in which the counit ε of the comonad is the counit ε of the adjunction, and comultiplication δ of the comonad is defined in terms of the unit by $\delta = F\eta G$.

A *coalgebra* (S, ℓ) for a comonad (D, ε, δ) on category $\mathbf{C}$ is an object S of $\mathbf{C}$ and an arrow, often called a 'coaction', $\ell : S \to DS$ satisfying the following two identities (also called counit and coassociativity):



A *morphism of coalgebras* from a coalgebra (S, ℓ) to a coalgebra (S', l') is an arrow $m : S \to S'$ making the following diagram commute:



3.2 Lenses are the coalgebras for the costate comonad

We revert to the argument order presented in Section 1 for the 'put' function, but curry it too, to yield $p : S \to (V \Rightarrow S)$; then we can tuple it with the 'get' function to yield a combination

$$\ell : S \to (V \Rightarrow S) \times V$$

That is, $\ell = \langle p, g \rangle$, and (interpreting ℓ, p, g as arrows in **set**) $\ell(s) = (p(s), g(s))$ for any s.

As with the algebraic model of lenses, we fix a view type V. This time we make use of the comonad $D = (\times V)(V \Rightarrow)$ arising from the cartesian closure adjunction $(\times V) \dashv (V \Rightarrow)$; on objects, D is given by

$$DA = (V \Rightarrow A) \times V$$

and on arrows,

$$Dh\,(f, v) = (h \cdot f, v)$$

Expressed pointwise, the counit and comultiplication operators of D are:

$$\varepsilon_A(f, v) = f(v)$$
$$\delta_A(f, v) = (\lambda u \to (f, u), v)$$

where $f : V \to A$.

The comonad (D, ε, δ) is sometimes called the 'costate comonad'—not because it has anything in particular to do with states, but because it is in some sense the dual of the 'state monad' $V \Rightarrow (A \times V)$ of computations operating on a hidden state of type V. O'Connor [O'C10] calls it the 'store comonad', because a value of type DS stores within it a value of type S that can be extracted via the counit ε. For the remainder of the paper, we will just refer to it as 'the comonad D'.

Rendered in the language of cartesian closed categories, the operations of the comonad D are:

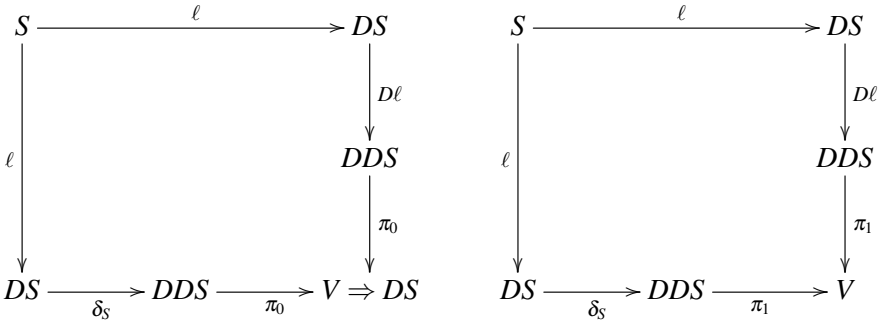
$$\begin{aligned} Dh &= ((V \Rightarrow h) \times V) \\ \delta &= \eta \times 1_V \end{aligned}$$

(and the counit ε of the comonad is just the counit ε of the adjunction, namely function application). In this language, the three laws of a very well-behaved lens $\ell = \langle p, g \rangle$ can be expressed as follows:

$$\begin{array}{lll} \varepsilon \cdot \langle p, g \rangle = 1_S & : S \to S & \text{-- get–put} \\ (V \Rightarrow) g \cdot p = \mathsf{const}_1 & : S \to (V \Rightarrow V) & \text{-- put–get} \\ (V \Rightarrow) p \cdot p = \mathsf{const}_0 \cdot p & : S \to (V \Rightarrow (V \Rightarrow S)) & \text{-- put–put} \end{array}$$

It turns out that these laws precisely coincide with the statement that $\ell : S \to DS$ is a coalgebra for the comonad D. To justify this claim, we show that ℓ makes the diagrams in Section 3.1 for counit and coassociativity of a coalgebra commute iff it is a very well-behaved lens.

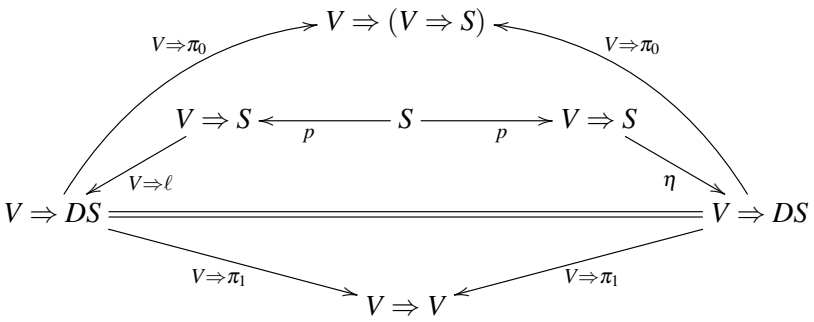
The counit triangle is clearly just a statement of the get–put law. The bottom right-hand corner DDS of the coassociativity square in Section 3.1 is a product object $(V \Rightarrow DS) \times V$, so proving that the square commutes amounts to showing that it does so for each projection:



The right-hand of these two squares is vacuous:

$$\begin{aligned} & \pi_1 \cdot D\ell \cdot \ell \\ = \quad & [\![\ Df = \ldots \times 1_V \ ]\!] \\ & 1_V \cdot \pi_1 \cdot \ell \\ = \quad & [\![\ \delta = \ldots \times 1_V \ ]\!] \\ & \pi_1 \cdot \delta \cdot \ell \end{aligned}$$

Let's look now at the left-hand of the two squares. Noting that the 'base' DS of the exponent in the bottom right-hand corner is again a product suggests looking at its two projections. The diagram below illustrates the situation:



The π_0 projections in the square have been promoted towards the upper left, using $\pi_0 \cdot D\ell \cdot \ell = (V \Rightarrow)\ell \cdot p$ and $\pi_0 \cdot \delta \cdot \ell = \eta \cdot p$. Then the square involving S, $V \Rightarrow S$ twice, and $V \Rightarrow DS$ has been rotated 45° clockwise, and the two copies of $V \Rightarrow DS$ separated; we have to show that this inner pentagon commutes. The two projections $V \Rightarrow (V \Rightarrow S)$ and $V \Rightarrow V$ of each copy of $V \Rightarrow DS$ have been identified; the inner product pentagon commutes iff the two hexagons rooted at S commute. The upper hexagon is equivalent to the put–put law:

$$
\begin{aligned}
& (V \Rightarrow)\pi_0 \cdot \eta \cdot p \\
= \quad & [\![\text{ definition }]\!] \\
& \mathsf{const}_0 \cdot p \\
= \quad & [\![\text{ put–put law }]\!] \\
& (V \Rightarrow)p \cdot p \\
= \quad & [\![\text{ lens }]\!] \\
& (V \Rightarrow)\pi_0 \cdot (V \Rightarrow)\ell \cdot p
\end{aligned}
$$

And the lower hexagon is equivalent to the put–get law:

$$
\begin{aligned}
& (V \Rightarrow)\pi_1 \cdot \eta \cdot p \\
= \quad & [\![\text{ definition }]\!] \\
& \mathsf{const}_1 \cdot p \\
= \quad & [\![\mathsf{const}_1 \text{ is a left zero of composition }]\!] \\
& \mathsf{const}_1 \\
= \quad & [\![\text{ put–get law }]\!] \\
& (V \Rightarrow)g \cdot p \\
= \quad & [\![\text{ lens }]\!] \\
& (V \Rightarrow)\pi_1 \cdot (V \Rightarrow)\ell \cdot p
\end{aligned}
$$

4 The correspondence

Now that we have established that every very well-behaved lens is both an algebra for a particular monad ($\Delta\Sigma$) and a coalgebra for a particular comonad (D), in this section we will explore the algebra-coalgebra correspondence.

4.1 Functoriality

We establish first that the correspondence extends to an isomorphism of categories.

Historically, lenses have mostly been studied individually, but recognising that in this paper we work with a fixed view V, and referring back to Section 1, it is clear what a 'morphism of lenses' should be: a morphism will be an arrow $S \to S'$ commuting with both the get arrows and the put arrows of the two lenses.

Referring back to Section 2, we see that an arrow m in the category $\mathbf{set}/V$ is a morphism of $\Delta\Sigma$-algebras if and only if it is a morphism of lenses. This follows since m is an arrow $g \to g'$ of $\mathbf{set}/V$ precisely when the function m (strictly Σm) satisfies $g'm = g$ in $\mathbf{set}$, and such an m is a morphism between $\Delta\Sigma$-algebras (g,p) and (g',p') precisely when it commutes with the respective put functions

$$\begin{array}{ccc} V \times S & \xrightarrow{\Delta\Sigma m} & V \times S' \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle p'} \\ S & \xrightarrow[m]{} & S' \end{array}$$

(suppressing the fibring over V—that is, the fact that each of the corners of that square really stands for an object in $\mathbf{set}/V$, and so is really an arrow from that object into V—by π_0, g and g' as appropriate).

Referring now to Section 3, an arrow $m : S \to S'$ in $\mathbf{set}$ is a morphism of D-coalgebras if and only if m is a morphism of lenses, since the commutativity of

$$\begin{array}{ccc} DS & \xrightarrow{Dm} & DS' \\ {\scriptstyle \ell}\uparrow & & \uparrow{\scriptstyle \ell'} \\ S & \xrightarrow[m]{} & S' \end{array}$$

amounts, with $DS' = (V \Rightarrow S') \times V$, $\ell = (p,g)$ and $\ell' = (p',g')$, to commutativity on each factor, and these are simply the commutativity of m with p and p' and with g and g' respectively.

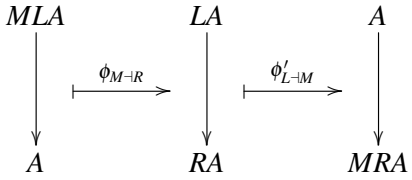
Thus in the case $\mathbf{C} = \mathbf{set}$ we have isomorphisms of categories between the category of lenses, the category of $\Delta\Sigma$-algebras, and the category of D-coalgebras.

Of course, we can say more. While lenses and their morphisms were defined over $\mathbf{set}$, the categories of algebras and coalgebras that we have been considering are defined, and remain equivalent, for arbitrary cartesian closed $\mathbf{C}$. Exploring this equivalence further is the topic of the next subsection.

4.2 The algebra-coalgebra correspondence

It is not especially unusual for the same objects to bear at once closely related algebra and coalgebra structures. As a non-trivial example consider the string of adjoints $L \dashv M \dashv R$ (where say $L,R : \mathbf{A} \to \mathbf{B}$ and $M : \mathbf{B} \to \mathbf{A}$). By virtue of the adjunctions, ML will be a monad and MR a comonad, and monad actions convert to comonad coactions by the transpositions $\phi_{M \dashv R}$ and $\phi'_{L \dashv M}$

of the two adjunctions:



In the situation we are considering in this paper we again have two adjoint pairs, but they are not as closely related as in the example just given, where the adjunctions share the functor M. Instead, the situation is as follows.

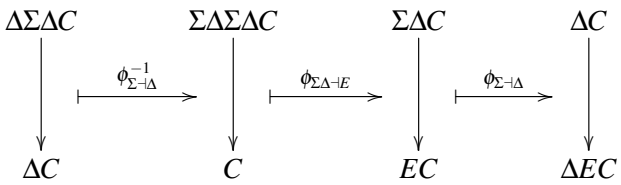
Suppose $\mathbf{C}$ is cartesian closed. Recall from Section 3 that the adjoint pair which generates the comonad D is the cartesian closure adjunction $(\times V) \dashv (V \Rightarrow)$. For convenience, denote the right adjoint E_V (exponentiation by V). Note that, ignoring the order of the factors in calculating products (which has been varied in the sections above to align with extant work), the left adjoint is just $\Sigma_V \Delta_V$. And as usual we will henceforward suppress the V subscripts since V remains fixed throughout this paper. Then our situation is as follows.

First, $\Sigma \dashv \Delta$ makes $\Delta\Sigma$ a monad. Furthermore $\Sigma\Delta$ is a comonad, but that is not important here. What is important is that that comonad has a right adjoint E, and this last adjunction yields the comonad $\Sigma\Delta E$. The previous subsection established (for the **set** case, but the arguments work in any cartesian closed category) the equivalence of categories

$$\Delta\Sigma\text{-algebras} \simeq \Sigma\Delta E\text{-coalgebras}.$$

So, rather than the simple $L \dashv M$ and $M \dashv R$ of our example above we have $\Sigma \dashv \Delta$ and (not Δ but) $\Sigma\Delta \dashv E$, and we seek to understand how actions for the monad generated by the first adjunction convert to coactions for the comonad generated by the second adjunction. (Note that we are not trying to establish again the equivalence. We just seek to isolate its generality, in so far as it follows from generalities about adjoints etc., and its particularities that might limit its occurrence to, for example, adjoints among categories $\mathbf{C}$ and $\mathbf{C}/V$.)

Now, actions for the monad have the form $\Delta\Sigma A \to A$, and there is no immediate transpose arising from the adjoints we are considering. However, supposing that V is inhabited, that is, that there exists in $\mathbf{C}$ an arrow $1 \to V$, then [JRW10] showed that Δ is monadic. We won't review monadicity here, but the importance for our application is that monadicity implies that every $\Delta\Sigma$-algebra is isomorphic to an algebra whose carrier is ΔC for some C in $\mathbf{C}$; that is, an algebra whose action is of the form $\Delta\Sigma\Delta C \to \Delta C$ (and that has important practical implications—in particular, it implies that very well behaved lenses have a particularly simple form known as 'constant complement'). Now we can begin to see a correspondence by transposing such an action under the two adjunctions:



(the steps using respectively $\Sigma \dashv \Delta$ from right to left, $\Sigma\Delta \dashv E$, and $\Sigma \dashv \Delta$).

Thus far the correspondence has been general and follows for any monadic Δ with left adjoint Σ such that $\Sigma\Delta$ has a right adjoint E. But having seen in Sections 2 and 3 the details of the encoding of the get function as either the carrier for the action or one factor in the coaction we can't expect to continue with general correspondences. Instead, we need to analyse the particularity of the arrow $\Delta C \to \Delta EC$ in $\mathbf{C}/V$.

Such an arrow is an arrow $V \times C \to V \times EC$ and so corresponds to two arrows $V \times C \to V$ and $V \times C \to EC$. But all objects in $\mathbf{C}/V$ of the form ΔX are fibred over V by π_0, so the first arrow $V \times C \to V$ must be simply π_0.

We obtain a coaction $\Sigma\Delta C \to V \times E\Sigma\Delta C = \Sigma\Delta E\Sigma\Delta C$ by tupling the arrow $\Sigma\Delta C = V \times C \to EC$ with the projection $\pi_0 : \Sigma\Delta C \to V$ and the constant at the identity $\Sigma\Delta C \to 1 \to EV$ (in which the last arrow is the transpose of the projection $V \times 1 \to V$ giving an arrow $1 \to (V \Rightarrow V)$). Conversely, given a coaction for $\Sigma\Delta E$ with carrier of the form $\Sigma\Delta C$ and whose first factor is a projection, if the coaction satisfies the put–get law then it decomposes as an arrow $\Sigma\Delta C \to EC$ along with a projection π_0 and a constant at the identity.

In summary, we start from $\Delta\Sigma$-actions, use monadicity of Δ to see that each such is isomorphic to a $\Delta\Sigma$-action on a carrier of the form ΔC for some C, and then have bijective correspondences from each such action by transpositions to an arrow $\Sigma\Delta C \to EC$, which in turn corresponds bijectively to D-coactions that satisfy the put–get law.

5 Conclusions

Overall, we have seen in detail that lenses, first defined as simple set-based structures with two operations, can be seen as algebras for a monad on $\mathbf{C}/V$ for $\mathbf{C}$ a category with products, and as coalgebras for a comonad on $\mathbf{C}$ for $\mathbf{C}$ cartesian closed, and we have analyzed in some detail the correspondence between these two (co)algebraic frameworks.

Of course the main value of the (co)algebraic analysis of useful structures is to be able to better understand and work with those structures and to find appropriate and useful generalizations of them. Much of that is ongoing work, but we make here some observations based on the results above.

To begin, the set-based definition of lenses includes three apparently equally important (if not equally controversial—see below) laws. Our analysis has revealed the special role played by one of those laws, the put–get law. In the coalgebraic approach that law arises, along with the put–put law, from the associativity of the coalgebra. But in the algebraic approach that law is needed to ensure that an action is indeed an arrow in $\mathbf{C}/V$ before we even consider whether the action is an action for an algebra or something less. And in the correspondence between actions and coactions, the put–get law is used just to complete the correspondence, again before considering what other laws (co)actions might satisfy. Thus there is an interesting asymmetry among the laws which does incidentally correspond to many practitioners' 'gut feelings': whatever else might result from completing a put with a new view state, it should be the case that that view is not altered as a result of the put.

Practitioners have also seen the laws as unequal in a different way. There has been considerable controversy over the put–put law, with examples being given of reasonable update strategies that cannot satisfy put–put [DXC11]. In the coalgebraic approach it is not clear how to separate

the put–put law from the put–get law, with both being embedded in the coassociativity square for a coalgebra, but in the algebraic approach the put–put law corresponds precisely to the associativity of the algebra, so studies are planned into pointed $\Delta\Sigma$-actions—those actions which satisfy the identity algebra law but not necessarily the associativity (put–put) law. An even more extreme generalization is available. Some workers study bare (co)actions that satisfy neither associativity nor identity identities (possibly even calling them (co)algebras when they are seen as being the fundamental structures). In the case of the actions studied here, such structures will still satisfy the put–get law, but possibly not the others, while in the case of the coactions none of the three lens laws need be satisfied.

Another kind of generalization has been implicit throughout this paper: lenses were defined in **set**, but the (co)algebraic versions treated here generalize to other categories with only a little extra structure (finite products, or cartesian closure). This generalization has been exploited already [JRW10] to capture and simplify the work of Hegner [Heg04], who studied ordered rather than discrete sets of states. Further generalizations are available immediately and are important—after all, states are rarely merely a set, but frequently have extra structure, forming a graph or a category for example. And it is useful that, depending on what kind of category those structures form (cartesian closed, or merely with finite products), we can choose to use the coalgebraic or algebraic approach.

There are thus various advantages in choosing one approach over the other (not least the fact that we have found that many computer scientists find the coalgebraic approach easier and more intuitive to use), and so there is definitely an advantage in having both approaches available. We offer just one more final example.

In the coalgebraic approach the carrier for the coalgebra is the domain of an arrow which embodies both the get and the put operations. In the algebra approach these two operations are separated—one arises because the carrier itself is an arrow, the get arrow, by virtue of being an object in $\mathbf{C}/V$, and the other, the put arrow, is the action. This means that it is possible to change the domain of the put arrow by changing the monad, while keeping the domain of the get arrow fixed. Johnson *et al.* [JRW12] argue that such a change is important because, when the states do have more structure than a mere set, the standard put domain $V \times S$ may be too big—it requires us to say how we would update, in set theoretic terms, any pair (v,s), even if the new view state v could never have been reached from the view of the state s. The resulting modified algebraic approach has interesting properties: it satisfies the put–put law, avoids many of the examples used in arguments against the put–put law, and extends to approaches which are not 'constant complement' [BS81]. This last is important, because it invalidates the mistaken belief that the presence of the put–put law implies constant-complement updating.

So, in summary, why conduct a mathematical investigation such as this? What useful developments are likely to follow? While it's too early to see what will be done with the coalgebraic approach, we can already cite some of the benefits of having the algebraic approach.

Indeed, mathematical analyses in general seek, among other things, to simplify existing work, to unify existing approaches, and to provide fruitful generalizations that can lead to new and broader applications of the work, while setting all this on a firm mathematical foundation. Instances of each of these are already becoming apparent for the algebraic approach. For simplification, the work of Hegner [Heg04] cited above included eight axioms which were shown to be codependent and reduced to the three. For unification, the original work on very well behaved

lenses, the work of Johnson *et al.* [JRW12], non-set-based cases such as Hegner's, and likely the work on delta-based lenses, can all now be seen as instances of algebras for a monad. And for fruitful generalisations, Johnson and Rosebrugh in this pre-proceedings volume use the algebraic approach to introduce refined put–put laws that have the same mathematical utility, but are satisfied in a much broader range of actual applications.

Acknowledgements: The observation that very well-behaved lenses are the coalgebras for the costate comonad is due to Russell O'Connor [O'C10]; he has applied this observation in design of a generic programming library called Multiplate [O'C11]. The development presented here in Section 3 was worked out at the *Dagstuhl Seminar on Bidirectional Transformations* in January 2011; we thank Schloß Dagstuhl for the convivial atmosphere so wonderfully conducive to scientific exchange and discovery.

This work was supported by the UK Engineering and Physical Sciences Research Council grant *Reusability and Dependent Types* (EP/G034516/1), and by the Australian Research Council.

Bibliography

[BS81] F. Bancilhon, N. Spyratos. Update Semantics of Relational Views. *ACM Transactions on Database Systems* 6:557–575, 1981.
doi:10.1145/319628.319634 1, 14

[CFH+09] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, J. F. Terwilliger. Bidirectional Transformations: A Cross-Discipline Perspective. In *Theory and Practice of Model Transformations (ICMT)*. Lecture Notes in Computer Science 5563, pp. 260–283. Springer, 2009.
doi:10.1007/978-3-642-02408-5_19 2

[DXC11] Z. Diskin, Y. Xiong, K. Czarnecki. From State- to Delta-Based Bidirectional Model Transformations: The Asymmetric Case. *Journal of Object Technology* 10:1–25, 2011.
doi:10.5381/jot.2011.10.1.a6 13

[FGM+07] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, A. Schmitt. Combinators for Bidirectional Tree Transformations: A Linguistic Approach to the View-Update Problem. *ACM Transactions on Programming Languages and Systems* 29(3):17, 2007.
doi:10.1145/1232420.1232424 1, 2

[FMW10] K. Fisher, Y. Mandelbaum, D. Walker. The Next 700 Data Description Languages. *Journal of the ACM* 57(2):1–51, 2010.
doi:10.1145/1667053.1667059 1

[Heg04] S. J. Hegner. An Order-Based Theory of Updates for Closed Database Views. *Annals of Mathematics and Artificial Intelligence* 40(1-2):63–125, 2004. doi:10.1023/A:1026158013113 14

[HPW11] M. Hofmann, B. Pierce, D. Wagner. Symmetric Lenses. In *Principles of Programming Languages*. Pp. 371–384. 2011. doi:10.1145/1926385.1926428 2

[JRW10] M. Johnson, R. Rosebrugh, R. J. Wood. Algebras and Update Strategies. *Journal of Universal Computer Science* 16(5):729–748, 2010. doi:10.3217/jucs-016-05-0729 3, 12, 14

[JRW12] M. Johnson, R. Rosebrugh, R. Wood. Lenses, Fibrations, and Universal Translations. *Mathematical Structures in Computer Science* 22:25–42, 2012. doi:10.1017/S0960129511000442 14, 15

[KP88] G. E. Krasner, S. T. Pope. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal of Object-Oriented Programming* 1(3):26–49, August/September 1988. 1

[ML71] S. Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer-Verlag, 1971. 3

[O'C10] R. O'Connor. Lenses are Exactly the Coalgebras for the Store Comonad. Nov. 2010. http://r6research.livejournal.com/23705.html. 3, 9, 15

[O'C11] R. O'Connor. Functor is to Lens as Applicative is to Biplate: Introducing MultiPlate. In *ACM Workshop on Generic Programming*. 2011. http://arxiv.org/abs/1103.2841. 15

[Pie91] B. C. Pierce. *Basic Category Theory for Computer Scientists*. MIT Press, 1991. 3

[Ste10] P. Stevens. Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions. *Software and System Modeling* 9(1):7–20, 2010. doi:10.1007/s10270-008-0109-9 2

[Voi09] J. Voigtländer. Bidirectionalization for Free! In *Principles of Programming Languages*. Pp. 165–176. 2009. doi:10.1145/1480881.1480904 2