Taking Updates Seriously*

Danel Ahman
Laboratory for Foundations of Computer Science, University of Edinburgh
10 Crichton Street, Edinburgh EH8 9LE, United Kingdom
d.ahman@ed.ac.uk
Tarmo Uustalu
Dept. of Software Science, Tallinn University of Technology
Akadeemia tee 21B, 12168 Tallinn, Estonia
tarmo@cs.ioc.ee

Abstract

We show how "taking updates seriously" leads from state-based lenses to update lenses and further; we witness a little hierarchy of types of lens that arises in a systematic way. Lenses of each type are characterized either as coalgebras of certain types of comonads or morphisms between certain types of comonads. In each case, a lens is simulation between two transition systems for suitable notions of transition system and simulation.

1 Introduction

In this paper, we discuss two types of asymmetric lens, update lenses of Ahman and Uustalu [AU14a] and update-update lenses (a new proposal of this paper). We show how state-based lenses, update lenses and update-update lenses arise systematically from the same paradigm: an asymmetric lens is about simulating changes in the view database by changes in the source database in a consistency-restoring manner; cf. McKinna's [McK16] slogan that symmetric lenses are bisimulations. Each subsequent notion of lens takes the idea of first-class changes (in our terminology, updates) more seriously than the previous one. In each case, a lens is essentially the same as a coalgebra of a comonad of an appropriate type or a morphism between two comonads of an appropriate type. We briefly compare update lenses and update-update lenses to the delta lenses of Diskin et al. [DXC11] and the category lenses of Johnson et al. [JRW12].

This paper is organized as follows. First, in Section 2, we recapitulate state-based lenses. In particular, we recall that state-based lenses are the same as coalgebras of costate comonads, which also happen to be the same as morphisms between costate comonads. Then, in Section 3, we introduce simply-typed update lenses as an improvement where changes to view states, or updates, are taken seriously, and show that they are the same as coalgebras of what could be called coupdate comonads. To allow every view state to have its own set of enabled updates, also applicable to the relevant source states, we introduce, in Section 4, the concept of a directed containers as the appropriate notion of transition system, and point out that every directed container determines

*The material of this paper was first presented in a talk at the NII Shonan Meeting on Bidirectional Transformations in Sept. 2016.

Copyright © by the paper's authors. Copying permitted for private and academic purposes.

In: R. Eramo, M. Johnson (eds.): Proceedings of the Sixth International Workshop on Bidirectional Transformations (Bx 2017), Uppsala, Sweden, April 29, 2017, published at http://ceur-ws.org

a dependently-typed coupdate comonad. We introduce dependently-typed update lenses as coalgebras of such comonads. In Section 5, we introduce a further generalization, called update-update lenses, where source states have their own enabled updates, and applying a view update to a source state goes via first translating it into a source update and then applying that. We show that update-update lenses are morphisms between coupdate comonads. In Section 6, we compare update-update lenses to delta lenses and categorical lenses. Finally, in Section 7, we briefly describe the related work, to conclude in Section 8.

Some additional material appears in appendices. In Appendix A, we briefly review comonads, comonad coalgebras and comonad morphisms. In Appendix B, we prove the bijection between coalgebras of a given comonad and morphisms from costate comonads to this comonad. Finally, in Appendix C, we define and compare cofunctors and opfibrations.

In the main part of the paper, we assume the reader to know basic category theory. While this includes the basics of comonads, we still give the main definitions in Appendix A for reference. We assume no knowledge about containers or directed containers, introducing them in Section 4, where they are first needed. We provide no proofs in the main part of the paper, but prove the central proposition in Appendix B.

2 State-Based Lenses

We will consider three types of asymmetric lens in this paper: state-based lenses, update lenses, and update-update lenses (the latter being an original contribution of this paper). They have a lot in common.

Any type of asymmetric lens is about two evolving databases, the source database and view (target) database. At any moment, both databases are in some state and the two states must be consistent, i.e., be in a certain relation. If the view state is changed, it must be possible to change also the source state in such a way that consistency is re-established. Reasonably, this notion of simulation of the view database by the source database should be compositional in the sense that i) no view change should induce no source change and ii) the composition of two view changes should induce the composition of the two induced source changes. In the types of lens that we consider, the two databases operate generally on two different sets of states, and the consistency relation is functional, in other words, the graph of a function extracting a view state from a source state.

The simplest type of lens we consider is state-based lenses in the sense of the very well-behaved lenses of Foster et al. [F+07].

The distinctive feature of state-based lenses is that the view state can be changed to any other view state and, moreover, the only thing identifying a view change (besides the current view state) is the new view state: view changes are unconstrained and, we might say, extensional. The same is true about source changes—they are likewise unconstrained and extensional—but of course a source change simulating a view change must re-establish consistency.

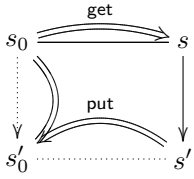
The official definition of state-based lenses is as follows.

Let S_0 and S be two sets (of source resp. view states). A *(very well-behaved) state-based lens* from S_0 to S is given by two maps $\mathsf{get} : S_0 \to S$ and $\mathsf{put} : S_0 \times S \to S_0$ such that

$$\mathsf{get}\,(\mathsf{put}\,(s_0, s')) = s'$$
$$s_0 = \mathsf{put}\,(s_0, \mathsf{get}\,s_0)$$
$$\mathsf{put}\,(\mathsf{put}\,(s_0, s'), s'') = \mathsf{put}\,(s_0, s'')$$

The map get is there to determine the view state corresponding to the current source state (defines the consistency relation between source and view states). The map put takes the current source state, the next view state and returns the next source state (defines simulation of view changes). The 1st equation asserts that consistency is restored in simulation. The 2nd and 3rd equations establish that simulation is compositional.

Pictorially, consistency and simulation are illustrated by the following diagram.



Simulation means that, given a source state s_0 and a view state s that are consistent ($\mathsf{get}\, s_0 = s$), any change of the view state, i.e., a new view state s', induces a change of the source state, i.e., a new source state $s_0' =_{\mathrm{df}} \mathsf{put}\,(s_0, s')$, so that consistency is re-established (we have $\mathsf{get}\, s_0' = \mathsf{get}\,(\mathsf{put}\,(s_0, s')) = s'$).

As an example, consider two versions of a simple bookshop database. The sets of source and view states are

$$S_0 =_{\mathrm{df}} \mathit{book} \Rightarrow \mathbb{N} \times \mathbb{N} \qquad S =_{\mathrm{df}} \mathit{book} \Rightarrow \mathbb{N}$$

The idea is that a source state is an association to every book (from some fixed list) of a price and a quantity (stock level); and a view state is an association to every book of just a price. The get operation corresponds to discarding the quantity of each book; the put operation changes the prices of all books leaving the quantities unchanged:

$$\mathsf{get}\, s_0 =_{\mathrm{df}} \lambda b.\, \mathsf{fst}\,(s_0\, b) \qquad \mathsf{put}\,(s_0, s) =_{\mathrm{df}} \lambda b.\, (s\, b, \mathsf{snd}\,(s_0\, b))$$

State-based lenses can be characterized in two ways in terms of costate comonads (a.k.a. array comonads), using comonad coalgebras or comonad morphisms. The coalgebraic characterization is due to Power and Shkaravska [PS04] and O'Connor [OCo11].

The *costate comonad* for a set S (of states) is the comonad defined by

$$DX =_{\mathrm{df}} S \times (S \Rightarrow X) \qquad \varepsilon_X\,(s, v) =_{\mathrm{df}} v\, s \qquad \delta_X\,(s, v) =_{\mathrm{df}} (s, \lambda s'.\, (s', \lambda s''.\, v\, s''))$$

First, there is a bijection between state-based lenses between S_0 and S, and coalgebras of the costate monad for S with S_0 as the carrier, i.e., a maps

$$\gamma : S_0 \to S \times (S \Rightarrow S_0)$$

satisfying some equations.

On the level of data, this bijection is only about straightforward packing of the data of a lens into one datum with currying and pairing. Given a coalgebra structure γ, we obtain the lens data by taking

$$\mathsf{get}\, s_0 =_{\mathrm{df}} \mathsf{fst}\,(\gamma\, s_0) \qquad \mathsf{put}\,(s_0, s') =_{\mathrm{df}} \mathsf{snd}\,(\gamma\, s_0)\, s'$$

Conversely, given a state-based lens $(\mathsf{get}, \mathsf{put})$, the coalgebra structure datum is

$$\gamma\, s_0 =_{\mathrm{df}} (\mathsf{get}\, s_0, \lambda s'.\, \mathsf{put}\,(s_0, s'))$$

But the lens equations also imply the coalgebra equations and vice versa.

Second, there is also a bijection between state-based lenses between S_0 and S, and comonad morphisms between the costate comonads for S_0 and S, i.e., natural transformations with components

$$\tau_X : S_0 \times (S_0 \Rightarrow X) \to S \times (S \Rightarrow X)$$

satisfying some equations.

The second characterization follows from the first thanks to a general fact that we also use later. For any comonad D, there is a bijection between coalgebras of the comonad D with set S_0 as the carrier, and comonad morphisms between the costate comonad for S_0 and the comonad D. Given a coalgebra structure $\gamma : S_0 \to D\, S_0$, the corresponding comonad morphism is given by $\tau_X\,(s_0, v) =_{\mathrm{df}} D\, v\,(\gamma s_0)$. Given a comonad morphism τ, the corresponding coalgebra structure is $\gamma\, s_0 =_{\mathrm{df}} \tau_{S_0}\,(s_0, \mathsf{id}_{S_0})$. We provide a full proof of this proposition in Appendix B.

3 Simply-Typed Update Lenses

We proceed to update lenses, as defined by Ahman and Uustalu [AU14a]. We first consider a simpler special case of the concept and then, in the next section, the fully general version.

As we saw, in a state-based lens, a view change is a just a move to a freely chosen next state. In an update lens, we are in a richer situation. There is a dedicated set of view updates; any view update is applicable to the current view state and gives the next view state. There is a distinguished trivial view update (which incurs

no update on the current view state) and any two view updates can be composed into a single view update (so that applying the composite view update to the current state has the same effect as applying first the first view update to the current state and then the second view update to the next state). Not every view state needs to be accessible from the current one by applying a view update and some view states may be accessed by several view updates. In other words, in contrast to state-based lenses, changes of view states are generally constrained and intensional. A move to a new view state must be simulatable by a move to a new source state so that consistency is restored. However, there is no separate set of source updates; one could say that view updates also double as source updates.

Update lenses are mathematically defined as follows.

We recall that a *monoid* is a set P with an element $\mathsf{o} : P$ and a map $\oplus : P \times P \to P$ such that

$$p \oplus \mathsf{o} = p$$
$$\mathsf{o} \oplus p = p$$
$$(p \oplus p') \oplus p'' = p \oplus (p' \oplus p'')$$

Recall also that a *right action* of a monoid $(P, \mathsf{o}, \oplus)$ on a set S is a map $\downarrow : S \times P \to S$ satisfying

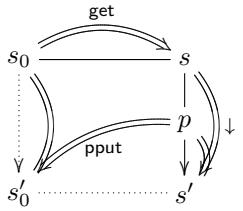
$$s \downarrow \mathsf{o} = s$$
$$s \downarrow (p \oplus p') = (s \downarrow p) \downarrow p'$$

Now, let S_0 be a set (of source states). Moreover, let S be another set (of view states), $(P, \mathsf{o}, \oplus)$ be a monoid (of view updates), and $\downarrow$ a right action of $(P, \mathsf{o}, \oplus)$ on S (application of a view update to a view state). A *(simply-typed) update lens* between S_0 and $(S, P, \downarrow, \mathsf{o}, \oplus)$ is given by two maps $\mathsf{get} : S_0 \to S$ and $\mathsf{pput} : S_0 \times P \to S_0$ such that

$$\mathsf{get}\,(\mathsf{pput}\,(s_0, p)) = \mathsf{get}\,s_0 \downarrow p$$
$$s_0 = \mathsf{pput}\,(s_0, \mathsf{o})$$
$$\mathsf{pput}\,(\mathsf{pput}\,(s_0, p), p') = \mathsf{pput}\,(s_0, p \oplus p')$$

The map pput describes how applying a view update to the current view state is simulated on the current source state.

Consistency and simulation in an update lens are illustrated by the following diagram.



Differently from the case of a state-based lens, in this picture, the new view state s' is not arbitrarily chosen; instead, it is obtained by applying an arbitrarily chosen view update p to the current view state s (i.e., $s' =_{\mathrm{df}} s \downarrow p$). The move from s to s' is simulated by a move from s_0 to s_0' where s_0' is defined by the map pput (i.e., $s_0' =_{\mathrm{df}} \mathsf{pput}\,(s_0, p)$). Again the role of the 1st equation is to assert that consistency is restored in simulation, and the 2nd and 3rd equations stipulate compositionality of simulation.

We can modify the bookshop example to obtain an update lens. Let $S_0 =_{\mathrm{df}} \mathit{book} \Rightarrow \mathbb{N} \times \mathbb{N}$ and $S =_{\mathrm{df}} \mathit{book} \Rightarrow \mathbb{N}$ as in the previous section. As the monoid of view updates we use the free monoid on the set $\mathit{book} \times \mathbb{Z}$, i.e.,

$$P =_{\mathrm{df}} (\mathit{book} \times \mathbb{Z})^* \qquad \mathsf{o} =_{\mathrm{df}} [] \qquad \oplus =_{\mathrm{df}} +\!\!+$$

so a view update is a sequence of price changes to books.

The get operation is defined as before. Application of view updates to view states and to source states is defined as follows.

$$\begin{array}{lll} s \downarrow [] & =_{\mathrm{df}} & v \\ s \downarrow ((b, c) :: p) & =_{\mathrm{df}} & (\lambda b'.\,\mathsf{if}\ b' = b\ \mathsf{then}\ \max(0, s\,b' + c)\ \mathsf{else}\ s\,b') \downarrow p \end{array}$$

$$\begin{array}{lll} \mathsf{pput}\,(s_0, []) & =_{\mathrm{df}} & v \\ \mathsf{pput}\,(s_0, (b, c) :: p) & =_{\mathrm{df}} & \mathsf{pput}\,(\lambda b'.\,\mathsf{if}\ b' = b\ \mathsf{then}\ (\max(0, \mathsf{fst}\,(s_0\,b') + c), \mathsf{snd}\,(s_0\,b'))\ \mathsf{else}\ s_0\,b', p) \end{array}$$

One might complain that the concept of update lens is a bit of an exaggeration, since, by the definition we have given, an update lens for $(S, P, \downarrow, \mathsf{o}, \oplus)$ is nothing but a $(P, \mathsf{o}, \oplus)$-set (S_0, pput) (i.e., a set S_0 and a right action pput of $(P, \mathsf{o}, \oplus)$ on S_0; this is stated by the 2nd and 3rd equations) together with a $(P, \mathsf{o}, \oplus)$-set morphism get between (S_0, pput) and $(S, \downarrow)$ (stated by the 1st equation). Yet, we claim that update lenses make enough practical sense and have enough theoretical significance to deserve attention, both in the special form we are discussing here as well as in the general form that we will consider in the next section.

One argument in defense of update lenses for theory is that, similarly to state-based lenses, update lenses are characterizable as comonad coalgebras and as comonad morphisms. The coalgebraic characterization is due to Ahman and Uustalu [AU14a].

Both characterizations use coupdate comonads. Any set S (of states), monoid $(P, \mathsf{o}, \oplus)$ (of updates) and right action $\downarrow$ of $(P, \mathsf{o}, \oplus)$ on S (application of an update to a state), define a comonad, which we call the *coupdate comonad* for $(S, P, \downarrow, \mathsf{o}, \oplus)$, by

$$D\,X =_{\mathrm{df}} S \times (P \Rightarrow X) \qquad \varepsilon_X\,(s, v) =_{\mathrm{df}} v\,\mathsf{o} \qquad \delta_X\,(s, v) =_{\mathrm{df}} (s, \lambda p.\,(s \downarrow p, \lambda p'.\,v\,(p \oplus p')))$$

We have: First, update lenses between S_0 and $(S, P, \downarrow, \mathsf{o}, \oplus)$ are in a bijection with coalgebras of the update comonad for $(S, P, \downarrow, \mathsf{o}, \oplus)$ with S_0 as the carrier.

Second, they are also in a bijection with comonad morphisms between the costate comonad for S_0 and the coupdate comonad for $(S, P, \downarrow, \mathsf{o}, \oplus)$. This an immediate consequence of the first characterization thanks to the same bijection between coalgebras of a given comonad and morphisms from costate comonads to this comonad that we exploited in the previous section.

A big difference of coupdate comonads from costate comonads is that coupdate comonads are compatible compositions of simpler comonads. Costate comonads admit no similar decomposition.

Given a set S, we have the *coreader comonad* defined by

$$D^0\,X =_{\mathrm{df}} S \times X \qquad \varepsilon^0_X\,(s, x) =_{\mathrm{df}} x \qquad \delta^0_X\,(s, x) =_{\mathrm{df}} (s, (s, x))$$

Given a monoid $(P, \mathsf{o}, \oplus)$, we have the *cowriter comonad* defined by

$$D^1\,X =_{\mathrm{df}} P \Rightarrow X \qquad \varepsilon^1_X\,v =_{\mathrm{df}} v\,\mathsf{o} \qquad \delta^1_X\,v =_{\mathrm{df}} \lambda p.\,\lambda p'.\,v\,(p \oplus p')$$

Right actions of $(P, \mathsf{o}, \oplus)$ on S are in a bijective correspondence with distributive laws of the comonad D^0 over the comonad D^1. As a consequence, coupdate comonads for $(S, P, \downarrow, \mathsf{o}, \oplus)$ are in a bijection with compatible compositions of the comonads D^0 and D^1.

This composite nature of coupdate comonads leads to a number of further characterizations of update lenses, e.g., as pairs of comonad coalgebras, comonad-monad bialgebras (pairs of a comonad coalgebra and a monad algebra) etc. Those are beyond the scope of this paper; we refer the interested reader to [AU14a] for a detailed account of them.

4 Dependently-Typed Update Lenses

In the type of update lenses that we considered in the previous section, view updates were always applicable and always composable. This may be unrealistic and is an unnecessary restriction. A generalization of update lenses from the previous section fixes this issue. Instead of a set, monoid and right action, we need to describe the view database in terms of a directed container.

Containers were introduced by Abbott et al. [AAG05] as a representation for a wide class of set functors (datatypes) in terms of sets and positions. Directed containers [ACU14] are containers with additional structure. They characterize those containers whose interpretation as a set functor carries a comonad structure.

A *container* is a set S (of shapes / or, in our application, states) and, for any $s : S$, a set $P\,s$ (of positions in the shape s / updates applicable to the state s)[1]. A *directed container* is a container (S, P) equipped with maps

- $\downarrow : (\Sigma s : S.\,P\,s) \to S$ (the subshape corresponding to a position in a shape / application of an update to a state),

[1] We originally adopted the letters S and P as mnemonics for *s*hapes and *p*ositions, but coincidentally they also work perfectly for *s*tates and *'p*dates.

- $\mathsf{o} : \Pi_{s:S}.\, P\, s$ (the root position / the trivial update), and
- $\oplus : \Pi_{s:S}.\, (\Sigma p : P\, s.\, P\, (s \downarrow p)) \to P\, s$ (translation of a position in a position's subshape / composition of two updates)

satisfying

$$s \downarrow \mathsf{o}_s = s$$
$$s \downarrow (p \oplus_s p') = (s \downarrow p) \downarrow p'$$
$$p \oplus_s \mathsf{o}_{s \downarrow p} = p$$
$$\mathsf{o}_s \oplus_s p = p$$
$$(p \oplus_s p') \oplus_s p'' = p \oplus_s (p' \oplus_{s \downarrow p} p'')$$

We notice that the data and equations of a directed container are like those of a set, monoid and a right action, modulo the presence of the "minor" (subscripted) arguments and the dependent typing. In particular, if $P\, s$, o_s, and $p \oplus_s p'$ do not actually depend on s, then we indeed have a set, monoid and right action.

A container (S, P) defines a set functor $[\![S, P]\!]^{\mathrm{c}} =_{\mathrm{df}} D$, called its *interpretation*, by

$$D\, X =_{\mathrm{df}} \Sigma s : S.\, P\, s \Rightarrow X$$

Given a directed container structure $(\downarrow, \mathsf{o}, \oplus)$ on the container, this functor obtains a comonad structure defined by

$$\varepsilon_X\, (s, v) =_{\mathrm{df}} v\, \mathsf{o}_s \qquad \delta_X\, (s, v) =_{\mathrm{df}} (s, \lambda p.\, (s \downarrow p, \lambda p'.\, v\, (p \oplus_s p')))$$

We call the comonad $[\![S, P, \downarrow, \mathsf{o}, \oplus]\!]^{\mathrm{dc}} =_{\mathrm{df}} (D, \varepsilon, \delta)$ the *interpretation* of $(S, P, \downarrow, \mathsf{o}, \oplus)$. We could also call it the *(dependently-typed) coupdate comonad* for $(S, P, \downarrow, \mathsf{o}, \oplus)$. Not unexpectedly, the dependently-typed concept is defined in exactly the same way as its simply-typed counterpart from the previous section, modulo the presence of the minor arguments and dependent typing.

Any comonad structure (ε, δ) on a set functor D that is the interpretation of some container (S, P) (i.e., $D\, X = \Sigma s : S.\, P\, s \Rightarrow X$) arises from a directed container structure $(\downarrow, \mathsf{o}, \oplus)$ on (S, P). In fact, directed container structures on (S, P) and comonad structures on D are in a bijection. Given a comonad structure (ε, δ), the corresponding directed container structure is defined by

$$\mathsf{o}_s =_{\mathrm{df}} \varepsilon_{P\, s}\, (s, \mathsf{id}) \qquad s \downarrow p =_{\mathrm{df}} \mathsf{fst}\, (\mathsf{snd}\, (\delta_{P\, s}\, (s, \mathsf{id}))\, p) \qquad p \oplus_s p' =_{\mathrm{df}} \mathsf{snd}\, (\mathsf{snd}\, (\delta_{P\, s}\, (s, \mathsf{id}))\, p)\, p'$$

Directed containers are in a bijection *up to isomorphism* with small categories. Given a directed container $(S, P, \downarrow, \mathsf{o}, \oplus)$, the corresponding small category is obtained as follows. The set of objects is S. The set of maps with domain $s : S$ is $P\, s$, which means that the total set of maps is $\bar{P} =_{\mathrm{df}} \Sigma s : S.\, P\, s$ and the domain of a map $(s, p) : \bar{P}$ is $\mathsf{src}\, p =_{\mathrm{df}} s$. The codomain of a map $(s, p) : \bar{P}$ is $\mathsf{tgt}\, p =_{\mathrm{df}} s \downarrow p$. The identity map on an object s is $\mathsf{id}_s =_{\mathrm{df}} (s, \mathsf{o}_s)$ and the 1st directed container equation ensures that its codomain is s, as required. A map (s, p) can only be composed with a map (s', p'), if $s \downarrow p = s'$, in which case the composition is $(s, p); (s', p') =_{\mathrm{df}} (s, p \oplus_s p')$. By the 2nd directed container equation the codomain of this map is $(s \downarrow p) \downarrow p' = s' \downarrow p'$, as required. The 3rd to the 5th equations ensure that composition is unital and associative.

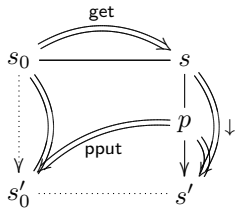
We are ready to define dependently-typed update lenses.

Let S_0 be a set (of source states). Let $(S, P, \downarrow, \mathsf{o}, \oplus)$ be a directed container (of view states and view updates, together with view update application). A *(dependently-typed) update lens* between S_0 and $(S, P, \downarrow, \mathsf{o}, \oplus)$ is given by two maps $\mathsf{get} : S_0 \to S$ and $\mathsf{pput} : (\Sigma s_0 : S_0.\, P\, (\mathsf{get}\, s_0)) \to S_0$ satisfying

$$\mathsf{get}\, (\mathsf{pput}\, (s_0, p)) = \mathsf{get}\, s_0 \downarrow p$$
$$s_0 = \mathsf{pput}\, (s_0, \mathsf{o}_{\mathsf{get}\, s_0})$$
$$\mathsf{pput}\, (\mathsf{pput}\, (s_0, p), p') = \mathsf{pput}\, (s_0, p \oplus_{\mathsf{get}\, s_0} p')$$

Again the only difference from the simply-typed concept of the previous section is the presence of minor arguments and dependent typing.

But exactly this difference makes the idea of updates enabled in a state work. Consider again the picture from the previous section. It is still valid.



Given a consistent pair of a current source state s_0 and view state s (i.e., $s = \mathsf{get}\, s_0$), view updates applicable to s come from the set $P\, s = P\, (\mathsf{get}\, s_0)$. The operation pput, which is supplied the current source state s_0 and an applicable view update p, must pick the next source state s'_0 consistently with the next view state $s' \downarrow p$. The 1st equation asserts that it does so.

We can modify the bookshop example from the previous section as follows to obtain a dependently-typed update lens. We define P as an inductive family by the rules

$$\frac{}{[] : P\, s} \qquad \frac{s\, b + c \geq 0 \quad p : P\, (\lambda b'.\, \mathsf{if}\ b' = b\ \mathsf{then}\ s\, b' + c\ \mathsf{else}\ s\, b')}{(b, c) :: p : P\, s}$$

The operations o and $\oplus$ remain defined essentially as before. Also $\downarrow$ and pput can be defined like before, but comparison with 0 is no longer necessary. A view update is only applicable to a view state, if the aggregate price change to each book is not too negative.

Not surprisingly, it remains true in the dependently-typed case that update lenses between S_0 and $(S, P, \downarrow, \mathsf{o}, \oplus)$ are in a bijection with coalgebras of the coupdate comonad for $(S, P, \downarrow, \mathsf{o}, \oplus)$ with S_0 as the carrier. And likewise they are in a bijection with comonad morphisms between the costate comonad for S_0 and the coupdate comonad for $(S, P, \downarrow, \mathsf{o}, \oplus)$ with S_0.

Let us now compare state-based and update lenses. Any set S can be canonically extended into a directed container $S^* =_{\mathrm{df}} (S, P, \downarrow, \mathsf{o}, \oplus)$ by choosing

$$P\, s =_{\mathrm{df}} S \qquad s \downarrow s' =_{\mathrm{df}} s' \qquad \mathsf{o}_s =_{\mathrm{df}} s \qquad s' \oplus_s s'' =_{\mathrm{df}} s''$$

In the directed container S^*, updates are just states and applying an update (state) to the current state means moving to that state. Viewed as small category, S^* is the *codiscrete category* with S as the set of objects, i.e., the category with exactly one map (s, s') between any two objects s and s'. In the next section, we shall see that S^* has a simple universal property.

It is easy to verify that the costate comonad for S is equal to the coupdate comonad for S^*.

As a result, coalgebras for the two comonads are on the nose the same thing, which in turn means that state-based lenses between S_0 and S, and update lenses between S_0 and S^* are essentially the same thing.

In other words, costate comonads are a special case of (dependently typed) coupdate comonads and state-based lenses are a special case of (dependently-typed) update lenses.

We note that this is not achievable with simply-typed coupdate comonads: there is no way to see the costate comonad for S as a simply-typed coupdate comonad, unless the set of view states S is a singleton. If S has multiple elements, we would need a different $\mathsf{o}_s = s$ for every $s : S$, and we are not allowed this in the simply-typed format.

But in one respect, we should point out, simply-typed coupdate comonads are more well-behaved than dependently-typed coupdate comonads: the former are compatible compositions of simpler comonads, but the latter are generally not. A decomposition is possible in terms of two relative comonads though.

5 Update-Update Lenses

In update lenses, we have first-class view updates, but no dedicated source updates. Instead, view updates double as source updates. This restriction can be lifted. We now proceed to the third type of lens of this paper, that we call update-update lenses. In an update-update lens, we have both view updates and source updates.

Simulation means finding, given a view update applicable to the current view state, a source update applicable to the current source state so that the next source state will be consistent with the next view state.

We will see that an update-update lens is exactly a morphism between directed containers.

We therefore begin by defining morphisms of containers and morphisms of directed containers.

A *container morphism* between two containers (S_0, P_0) and (S, P) is given by maps $t : S_0 \to S$ (the shape map) and $q : \Pi_{s_0 : S_0}.\, P\,(t\,s_0) \to P_0\,s_0$ (the position map). A *directed container morphism* between two directed containers $(S_0, P_0, \downarrow_0, \mathsf{o}_0, \oplus_0)$ and $(S, P, \downarrow, \mathsf{o}, \oplus)$ is a morphism (t, q) between the underlying containers satisfying

$$t\,(s_0 \downarrow_0 q_{s_0}\,p)) = t\,s_0 \downarrow p$$
$$\mathsf{o}_{0\,s_0} = q_{s_0}\,\mathsf{o}_{t\,s_0}$$
$$q_{s_0}\,p \oplus_{0\,s_0} q_{s_0 \downarrow_0 q_{s_0}\,p}\,p' = q_{s_0}\,(p \oplus_{t\,s_0} p')$$

A morphism (t, q) between containers (S_0, P_0) and (S, P) defines a natural transformation $[\![t, q]\!]^{\mathrm{c}} =_{\mathrm{df}} \tau$ between their interpretations $[\![S_0, P_0]\!]^{\mathrm{c}} =_{\mathrm{df}} D_0$ and $[\![S, P]\!]^{\mathrm{c}} =_{\mathrm{df}} D$ (the *interpretation* of (t, q)) by

$$\tau_X(s_0, v) =_{\mathrm{df}} (t\,s_0, \lambda p.\, v\,(q_{s_0}\,p))$$

If (t, q) is a morphism between directed containers $(S_0, P_0, \downarrow_0, \mathsf{o}_0, \oplus_0)$ and $(S, P, \downarrow, \mathsf{o}, \oplus)$, then τ is a comonad morphism between $[\![S_0, P_0, \downarrow_0, \mathsf{o}_0, \oplus_0]\!]^{\mathrm{dc}} =_{\mathrm{df}} (D_0, \varepsilon_0, \delta_0)$ and $[\![S, P, \downarrow, \mathsf{o}, \oplus]\!]^{\mathrm{dc}} =_{\mathrm{df}} (D, \varepsilon, \delta)$; we define $[\![t, q]\!]^{\mathrm{dc}} =_{\mathrm{df}} \tau$.

Any natural transformation τ between the interpretations D_0 and D of two containers (S_0, P_0) and (S, P) is an interpretation of a unique container morphism, namely (t, q) where

$$t\,s_0 =_{\mathrm{df}} \mathsf{fst}\,(\tau_{P\,s_0}(s_0, \lambda p_0.\, p_0)) \qquad q_{s_0}\,p =_{\mathrm{df}} \mathsf{snd}\,(\tau_{P\,s_0}(s_0, \lambda p_0.\, p_0))\,p$$

If τ is a comonad morphism between the interpretations $(D_0, \varepsilon_0, \delta_0)$ and (D, ε, δ) of two directed containers $(S_0, P_0, \downarrow_0, \mathsf{o}_0, \oplus_0)$ and $(S, P, \downarrow, \mathsf{o}, \oplus)$, then (t, q) is a directed container morphism interpreting to τ.

Containers and containers morphisms form a monoidal category **Cont**, interpretation of containers is a fully faithful monoidal functor from **Cont** to $[\mathbf{Set}, \mathbf{Set}]$. Directed containers and directed container morphisms form a category **DCont**, interpretation of directed containers is fully faithful functor from **DCont** to **Comonad**(**Set**). In fact, **DCont** is isomorphic to the category **Comonoid**(**Cont**) and is the pullback in **CAT** of $U : \mathbf{Comonad}(\mathbf{Set}) \to [\mathbf{Set}, \mathbf{Set}]$ along $[\![-]\!]^{\mathrm{c}} : \mathbf{Cont} \to [\mathbf{Set}, \mathbf{Set}]$.

While directed containers are in a bijection up to isomorphism with small categories, the category of directed containers is not equivalent to the category of small categories. Directed container morphism are not at all like functors between small categories, they are quite different. They turn out to map to what Aguiar [Agu97] termed cofunctors, but with the source and target categories switched. We give the definition.

A *cofunctor* between small categories $(S, \bar{P}, \mathsf{src}, \mathsf{tgt}, \mathsf{id}, ;)$ and $(S_0, \bar{P}_0, \mathsf{src}_0, \mathsf{tgt}_0, \mathsf{id}_0, ;_0)$ is given by two maps $t : S_0 \to S$ (the object map) and $\bar{q} : (\Sigma s_0 : S_0.\, \Sigma p : \bar{P}.\, t\,s_0 = \mathsf{src}\,p) \to \bar{P}_0$ (the morphism map) satisfying $\mathsf{src}_0\,(\bar{q}\,(s_0, p)) = s_0$ and

$$t\,(\mathsf{tgt}_0\,(\bar{q}\,(s_0, p))) = \mathsf{tgt}\,p$$
$$\mathsf{id}_{0\,s_0} = \bar{q}\,(s_0, \mathsf{id}_{t\,s_0})$$
$$\bar{q}\,(s_0, p)\,;_0\,\bar{q}\,(\mathsf{tgt}_0\,(\bar{q}\,(s_0, p)), p') = \bar{q}\,(s_0, p\,;\,p')$$

While a functor maps objects and maps of the source category to those in the target category, a cofunctor's object map is from the target category to the source category, but the morphism map is still from the source to the target category. A cofunctor looks a bit like an opcleavage, but it is not one. We will comment on the exact differences in the next section.

The category **DCont** of directed containers is equivalent to the opposite category of the category $\overleftarrow{\mathbf{Cat}}$ of small categories and cofunctors. Given a directed container morphism (t, q), the corresponding cofunctor is $(t, \bar{q})$ where $\bar{q}$ is defined by $\bar{q}\,(s_0, (t\,s_0, p)) =_{\mathrm{df}} (s_0, q_{s_0}\,p)$.

In the previous section, we discussed the construction of the directed container S_0^* from a set S_0. The corresponding small category was the codiscrete category on S_0. Now that we have fixed what we want to consider as morphisms between directed containers, we can say that this directed container is the free directed container on S_0, or, as a small category, the free object in $(\overleftarrow{\mathbf{Cat}})^{\mathrm{op}}$ on S_0. It is also at the same time the cofree small category (in the sense of being the cofree object in **Cat**) on S_0. (Note that in the first case, we consider small categories and cofunctors, in the second case, small categories and functors.)

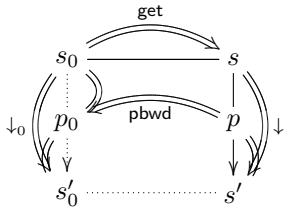
We are all set to define update-update lenses.

An *update-update lens* between two directed containers $(S_0, P_0, \downarrow_0, \mathsf{o}_0, \oplus_0)$ and $(S, P, \downarrow, \mathsf{o}, \oplus)$ (for source states, updates and update application resp. view states, updates and update application) is given by two maps $\mathsf{get} : S_0 \to S$ and $\mathsf{pbwd} : \Pi_{s_0:S_0}.\ P\,(\mathsf{get}\,s_0) \to P_0\,s_0$ such that

$$\mathsf{get}\,(s_0 \downarrow_0 \mathsf{pbwd}_{s_0}\,p)) = \mathsf{get}\,s_0 \downarrow p$$
$$\mathsf{o}_{0\,s_0} = \mathsf{pbwd}_{s_0}\,\mathsf{o}_{\mathsf{get}\,s_0}$$
$$\mathsf{pbwd}_{s_0}\,p \oplus_{0\,s_0} \mathsf{pbwd}_{s_0 \downarrow_0 \mathsf{pbwd}_{s_0}\,p}\,p' = \mathsf{pbwd}_{s_0}\,(p \oplus_{\mathsf{get}\,s_0} p')$$

In other words, an update-update lens is just a directed container morphism, we only renamed t to get and q to pbwd. Or, we could also say, it is a cofunctor.

Consistency and simulation in an update-update lens are illustrated in this picture.



If the current source and view state are s_0 and $s =_{\mathrm{df}} \mathsf{get}\,s_0$, then a view update p applicable to s is simulated by an update $p_0 =_{\mathrm{df}} \mathsf{pbwd}_{s_0}\,p$ applicable to s_0. The next view state $s' =_{\mathrm{df}} s \downarrow p$ must be consistent with the next source state $s'_0 =_{\mathrm{df}} s_0 \downarrow_0 p_0$ and this is stipulated by the 1st equation.

There are a many ways to modify the bookshop example to yield an update-update lens. Recall that in the previous sections we had $S_0 =_{\mathrm{df}} \mathit{book} \Rightarrow \mathbb{N} \times \mathbb{N}$ and $S =_{\mathrm{df}} \mathit{book} \Rightarrow \mathbb{N}$, with a source state associating to every book a price and quantity, and a view state only a price. Here are some possibilities.

(i) Take $P\,s =_{\mathrm{df}} (\mathit{book} \times \mathbb{Z})^*$ with the intent that a view update is an (unsafe) sequence of book and price change pairs, and take $P_0\,s_0 =_{\mathrm{df}} (\mathit{book} \times (\mathbb{Z} + \mathbb{Z}))^*$ with the intent that a source update is an association of an (unsafe) price or quantity change for every book. The function pbwd would be the obvious embedding of $P\,(\mathsf{get}\,s_0)$ in $P_0\,s_0$.

$$\begin{array}{lll} \mathsf{pbwd}\,[] & =_{\mathrm{df}} & [] \\ \mathsf{pbwd}\,((b,c) :: p) & =_{\mathrm{df}} & ((b, \mathsf{inl}\,c) :: \mathsf{pbwd}\,p \end{array}$$

(ii) Let P remain as in (i), but take $P_0\,s_0 =_{\mathrm{df}} \mathit{book} \Rightarrow \mathbb{Z} \times \mathbb{Z}$, with the idea that a source update is an association of a price change and quantity change to every book. The function pbwd should then aggregate the price changes for every book by summing them up; the quantity change of every book should be 0.

$$\begin{array}{lll} \mathsf{pbwd}\,[] & =_{\mathrm{df}} & \lambda b'.\,(0,0) \\ \mathsf{pbwd}\,((b,c) :: p) & =_{\mathrm{df}} & \lambda b'.\,(\mathsf{if}\ b' = b\ \mathsf{then}\ \mathsf{fst}\,(\mathsf{pbwd}\,p\,b') + c\ \mathsf{else}\ \mathsf{fst}\,(\mathsf{pbwd}\,p\,b'), 0) \end{array}$$

(iii) Let P_0 remain as in (i), but take $P\,s =_{\mathrm{df}} (\mathit{book} \times \mathbb{Z}_{<0})^*$, so a view update is a sequence of book and negative price change pairs. pbwd remains essentially as in (i).

Directed container theory tells us that update-update lenses between $(S_0, P_0, \downarrow_0, \mathsf{o}_0, \oplus_0)$ and $(S, P, \downarrow, \mathsf{o}, \oplus)$ are in a bijection with the comonad morphisms between the corresponding two coupdate comonads. There no characterization of update-update lenses with comonad coalgebras similar to those of state-based lenses or update lenses, because the first comonad is a coupdate comonad and not a costate comonad.

Because the costate comonad for a set S_0 equals the coupdate comonad for the free directed container S_0^*, update lenses between S_0 and $(S, P, \downarrow, \mathsf{o}, \oplus)$ are the same thing as update-update lenses between S_0^* and $(S, P, \downarrow, \mathsf{o}, \oplus)$. In this way, update lenses are a special case of update-update lenses.

Update lenses can be seen as a special case of update-update lenses also in another way. Given an update lens $(\mathsf{get}, \mathsf{pput})$ between S_0 and $(S, P, \downarrow, \mathsf{o}, \oplus)$, we can equip S_0 with a directed container structure by

$$P\,s_0 =_{\mathrm{df}} P\,(\mathsf{get}\,s_0) \qquad s_0 \downarrow_0 p_0 =_{\mathrm{df}} \mathsf{pput}\,(s_0, p_0) \qquad \mathsf{o}_{0\,s_0} =_{\mathrm{df}} \mathsf{o}_{\mathsf{get}\,s_0} \qquad p_0 \oplus_{0\,s_0} p'_0 =_{\mathrm{df}} p_0 \oplus_{\mathsf{get}\,s_0} p'_o$$

We then have the update-update lens $(\mathsf{get}, \mathsf{pbwd})$ where $\mathsf{pbwd}_{s_0}\,p =_{\mathrm{df}} p$. This construction substantiates the intuition that, in an update lens, view updates double as source updates.

6 Comparison to Delta Lenses and Category Lenses

Let us compare very briefly the update-update lenses introduced in the previous section to Diskin et al.'s delta lenses [DXC11] and Johnson et al.'s category lenses [JRW12].

Common to all three is that there are not only view updates, but also source updates. A big difference is that update-update lenses have only backward-pushing of updates (view updates are mapped to source updates, depending on the current source state). In delta and category lenses, updates can also be pushed forward; there is a map sending source updates to view updates.

For the sake of comparison, we present the definitions of delta and category lenses in terms of directed containers.

Given two directed containers $(S_0, P_0, \downarrow_0, \mathsf{o}_0, \oplus_0)$ and $(S, P, \downarrow, \mathsf{o}, \oplus)$ (for source states, updates and update application, resp. view states, updates and update application). A *delta lens* is given by maps $\mathsf{get} : S_0 \to S$, $\mathsf{pfwd} : \Pi_{s_0:S_0}.\ P_0\, s_0 \to P\,(\mathsf{get}\, s_0)$, and $\mathsf{pbwd} : \Pi_{s_0:S_0}.\ P\,(\mathsf{get}\, s_0) \to P_0\, s_0$ satisfying

$$\begin{gathered}
\mathsf{get}\, s_0 \downarrow \mathsf{pfwd}_{s_0}\, p_0 = \mathsf{get}\,(s_0 \downarrow_0 p_0) \\
\mathsf{o}_{\mathsf{get}\, s_0} = \mathsf{pfwd}_{s_0}\, \mathsf{o}_{0\, s_0} \\
\mathsf{pfwd}_{s_0}\, p_0 \oplus_{\mathsf{get}\, s_0} \mathsf{pfwd}_{s_0 \downarrow_0 p_0}\, p_0' = \mathsf{pfwd}_{s_0}\,(p_0 \oplus_{0\, s_0} p_0') \\
\mathsf{get}\,(s_0 \downarrow_0 \mathsf{pbwd}_{s_0}\, p) = \mathsf{get}\, s_0 \downarrow p \\
\mathsf{o}_{0\, s_0} = \mathsf{pbwd}_{s_0}\, \mathsf{o}_{\mathsf{get}\, s_0} \\
\mathsf{pbwd}_{s_0}\, p \oplus_{0\, s_0} \mathsf{pbwd}_{s_0 \downarrow_0 \mathsf{pbwd}_{s_0}\, p}\, p' = \mathsf{pbwd}_{s_0}\,(p \oplus_{\mathsf{get}\, s_0} p') \\
\mathsf{pfwd}_{s_0}\,(\mathsf{pbwd}_{s_0}\, p) = p
\end{gathered}$$

(The 2nd and 4th equation here are in fact derivable from the others.) It is immediate from this definition, that a delta lens is an update-update lens with additional structure. pfwd simulates a source update by a view update. The 7th equation says that simulation of a view update p by a source update p_0 by pbwd must be "correct" in the sense that pfwd must simulate p_0 by the same view update p.

In terms of small categories, we do not only have a cofunctor $(\mathsf{get}, \mathsf{pbwd})$ from the target category to the source category in a delta lens, but also a functor $(\mathsf{get}, \mathsf{pfwd})$ from the source category to the target category, with the same object mapping get. Composition of pfwd after pbwd must be identity. A structure like this could be called a splitting pre-opcleavage (where we say 'pre-' to express that have waived the standard opCartesianness requirement on the lifts $\mathsf{pbwd}_{s_0}\, p$).

A *category lens* is a delta lens with an additional map

$$\mathsf{fill} : \Pi_{s_0:S_0}.\ \Pi_{p:P\,(\mathsf{get}\, s_0)}.\ (\Sigma p_0 : P_0\, s_0.\ \Sigma p' : P\,(\mathsf{get}\, s_0 \downarrow p).\ \mathsf{pfwd}_{s_0}\, p_0 = p \oplus_{\mathsf{get}\, s_0} p') \to P_0\,(s_0 \downarrow_0 \mathsf{pbwd}_{s_0}\, p)$$

satisfying

$$\begin{gathered}
\mathsf{pbwd}_{s_0}\, p \oplus_{0\, s_0} \mathsf{fill}_{s_0,p}\,(p_0, p') = p_0 \\
\mathsf{pfwd}_{s_0 \downarrow_0 \mathsf{pbwd}_{s_0}\, p}\,(\mathsf{fill}_{s_0,p}\,(p_0, p')) = p' \\
\mathsf{fill}_{s_0,p}\,(\mathsf{pbwd}_{s_0}\, p, \mathsf{o}_{\mathsf{get}\, s_0 \downarrow p}) = \mathsf{o}_{0\, s_0 \downarrow_0 \mathsf{pbwd}_{s_0}\, p} \\
\mathsf{fill}_{s_0,p}\,(p_0 \oplus_{0\, s_0} p_0', p' \oplus_{\mathsf{get}\, s_0 \downarrow p} \mathsf{pfwd}_{s_0 \downarrow p_0}\, p_0') = \mathsf{fill}_{s_0,p}\,(p_0, p') \oplus_{0\, s_0 \downarrow_0 \mathsf{pbwd}_{s_0}\, p} p_0'
\end{gathered}$$

Intuitively, in a category lens, simulation of a view update by a source update causes least change to the source state. Suppose we are simulating a view update $p : P\,(\mathsf{get}\, s_0)$ consistent with some source state $s_0 : S_0$. The simulating source update is $\mathsf{pbwd}_{s_0}\, p : P_0\, s_0$. Suppose we have some other source update $p_0 : P_0\, s_0$ for the same source state s_0 whose view simulation $\mathsf{pfwd}_{s_0}\, p_0 : P\,(\mathsf{get}\, s_0)$ factors through p, i.e., $\mathsf{pfwd}_{s_0}\, p_0 = p \oplus p'$ for some p'. Then, on the source side, p_0 factors through $\mathsf{pbwd}_{s_0}\, p$, as we have $\mathsf{pbwd}_{s_0}\, p \oplus_0 \mathsf{fill}_{s_0,p}\,(p_0, p') = p_0$, moreover this is the unique such factoring of p_0.

In terms of small categories, a category lens is a splitting opcleavage. The additional data and equations assure that the lifts $\mathsf{pbwd}_{s_0}\, p$ are opCartesian.

We would like to argue that although pfwd can serve as a kind of quality yardstick of pbwd, it can often be unnatural from the applications viewpoint to require the map pfwd. It is easy to construct meaningful update-update lenses that are not delta lenses. The presence of an operation pfwd forces that all source updates, even those not in the range of pbwd, must be simulable as view updates, which makes it impossible to accommodate

example (iii) from the previous section: the equation $\mathsf{get}\, s_0 \downarrow \mathsf{pfwd}_{s_0}\, p_0 = \mathsf{get}\,(s_0 \downarrow_0 p_0)$ cannot be met. The equation $\mathsf{pfwd}_{s_0}\,(\mathsf{pbwd}_{s_0}\, p) = p$ can only hold when pbwd is injective, which rules out example (ii) from the previous section.

But any update lens can be considered as a delta lens, by using its view updates also as source updates (i.e., $P_0\, s_0 =_{\mathrm{df}} P\,(\mathsf{get}\, s_0)$, $\mathsf{pbwd}_{s_0}\, p =_{\mathrm{df}} p$, and $s_0 \downarrow_0 p =_{\mathrm{df}} \mathsf{pput}\,(s_0, p)$). It is then also a category lens.

In contrast, when we consider the source states of an update lens as its source updates (i.e., $P_0\, s_0 =_{\mathrm{df}} S_0$, $\mathsf{pbwd}_{s_0}\, p =_{\mathrm{df}} \mathsf{pput}\,(s_0, p)$, and $s_0 \downarrow_0 s_0' =_{\mathrm{df}} s_0'$), then the update lens is generally not a delta lens. Indeed, since, for any s_0, s_0', we have $s_0 \downarrow_0 s_0' = s_0'$, we must also have $\mathsf{get}\, s_0 \downarrow \mathsf{pfwd}_{s_0}\, s_0' = \mathsf{get}\, s_0'$. But there may well exist source states s_0, s_0' for which there is no view update $p : P\,(\mathsf{get}\, s_0)$ satisfying $\mathsf{get}\, s_0 \downarrow p = \mathsf{get}\, s_0'$.

On the theoretical side, state-based lenses, update lenses, and update-update lenses admit concise characterizations as comonad morphisms. We do not know whether something similar is also achievable for delta or category lenses.

7 Related Work

The literature on lenses has grown quite large. As this paper is on notions of asymmetric lens, where one distinguishes between a source database and a view (target) database, we only discuss works on those, to explain how the different strands of study developed and cross-fertilized.

State-based lenses were introduced by Foster et al. [F+07]. Two finer concepts of lens relying on first-class state changes or deltas, namely, delta lenses and categorical lenses, were introduced independently by Diskin et al. [DXC11] and Johnson et al. [JRW12]. Johnson and Rosebrugh [JR13] worked out the interrelationship of these two concepts. To be able to compare the different types of asymmetric lens to Hofmann et al.'s [HPW12] on (symmetric) edit lenses, Johnson and Rosebrugh [JR16] introduced a yet different variation, asymmetric edit lenses.

Power and Shkaravska [PS04] and O'Connor [OCo11] were probably the first to notice that state-based lenses are the same as coalgebras of costate comonads. Johnson et al. [JRW10] at the same time promoted an algebraic characterization of state-based lenses. Gibbons and Johnson [GJ12] explained why both are possible by showing that they are directly interderivable.

Containers as a useful "syntax" for a wide class of set functors were introduced by Abbott et al. [AAG05]. Directed containers as characterization of those containers whose endofunctor interpretation carries a comonad structure were introduced by Ahman et al. [ACU14]. In a later work [AU16], they noticed that directed containers are the same as categories whereas morphisms between them are not functors, but particular "relative splitting pre- opcleavages". Now they know that this concept of map between two categories is 20 years old and was introduced under the name of cofunctor by Aguiar [Agu97]. Ahman and Uustalu [AU14b] noticed that, in addition to the interpretation as comonads, directed containers can also be interpreted ("cointerpreted") as monads of a particular type, which they called update monads, generalizing state monads. Only subsequently [AU14a], they noticed that coalgebras of coupdate comonads (comonads denoted by a directed container) make a useful notion of lens, which they termed update lenses. They also showed that, similarly to state-based lenses, update lenses admit multiple characterizations, among them characterizations as algebras.

8 Conclusion

We hope to have demonstrated in this paper that both update lenses and update-update lenses are natural concepts. Both are fit for their application purpose and well-motivated theoretically, which is witnessed in particular by the fact that they can be characterized in several canonical ways. The leading intuition should in both cases be that a lens is a simulation between two transition systems, the difference being between whether the simulating system must use the same alphabet of transition labels as the simulated system or can have its own. Update lenses are the same as coalgebras of a comonad whose underlying endofunctor is the interpretation of a container, update-update lenses are morphisms between two such comonads.

We compared update-update lenses to delta and categorical lenses, singled out the differences, and argued that update-update lenses have both practical and theoretical advantages.

Acknowledgements

Tarmo Uustalu is grateful to Michael Johnson and Paul-André Melliès for discussions.

This research was supported by the Estonian Ministry of Education and Research institutional research grant No. IUT33-13.

References

[AAG05] M. Abbott, T. Altenkirch, N. Ghani. Containers: constructing strictly positive types. *Theor. Comput. Sci.*, v. 342, n. 1, pp. 3–27, 2005. doi: 10.1016/j.tcs.2005.06.002

[Agu97] M. Aguiar. *Internal Categories and Quantum Groups*. Cornell University, 1997. `http://www.math.cornell.edu/~maguiar/thesis2.pdf`

[ACU14] D. Ahman, J. Chapman, T. Uustalu. When is a container a comonad? *Log. Methods in Comput. Sci.*, v. 10, n. 3, article 14, 2014. doi: 10.2168/lmcs-10(3:14)2014

[AU14a] D. Ahman, T. Uustalu. Coalgebraic update lenses. In B. Jacobs, A. Silva, S. Staton, eds., *Proc. of 30th Conf. on Mathematical Foundations of Programming Semantics, MFPS XXX (Ithaca, NY, June 2014)*, v. 308 of *Electron. Notes in Theor. Comput. Sci.*, pp. 25–48. Elsevier, 2014. doi: 10.1016/j.entcs.2014.10.003

[AU14b] D. Ahman, T. Uustalu. Update monads: cointerpreting directed containers. In R. Matthes, A. Schubert, eds., *Proc. of 19th Conf. on Types for Proofs and Programs, TYPES 2013 (Toulouse, Apr. 2013)*, v. 26 of *Leibniz Int. Proc. in Inform.*, pp. 1–23. Dagstuhl Publishing, 2014. doi: 10.4230/lipics.types.2013.1

[AU16] D. Ahman, T. Uustalu. Directed containers as categories. In R. Atkey, N. Krishnaswami, eds., *Proc. of 6th Wksh. on Mathematically Structured Functional Programming, MSFP 2016 (Eindhoven, April 2016)*, v. 207 of *Electron. Proc. in Theor. Comput. Sci.*, pp. 89–98. Open Publishing Assoc., 2016. doi: 10.4204/eptcs.207.5

[DXC11] Z. Diskin, Y. Xiong, K. Czarnecki. From state- to delta-based bidirectional model transformations: the asymmetric ase. *J. of Object Technol.*, v. 10, article 6, 2011. doi: 10.5381/jot.2011.10.1.a6

[F+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 Trans. on Program. Lang. and Syst.*, v. 29, n. 3, article 17, 2007. doi: 10.1145/1232420.1232424

[GJ12] J. Gibbons, M. Johnson. Relating algebraic and coalgebraic descriptions of lenses. In F. Hermann, J. Voigtländer, eds., *Proc. of 1st Int. Wksh. on Bidirectional Transformations, BX 2012 (Tallinn, March 2012)*, v. 49 of *Electron. Commun. of EASST*, 16 pp. 2012. doi: 10.14279/tuj.eceasst.49.726

[HPW12] M. Hofmann, B. C. Pierce, D. Wagner. Edit lenses. In *Proc. of 39th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, POPL '12 (Philadelphia, PA, Jan. 2012)*, pp. 495–508. ACM, 2012. doi: 10.1145/2103621.2103715

[JR13] M. Johnson, R. Rosebrugh. Delta lenses and opfibrations. In P. Stevens, J. F. Terwilliger, eds., *Proc. of 2nd Int. Wksh. on Bidirectional Transformations, BX 2013 (Rome, March 2013)*, v. 57 of *Electron. Commun. of EASST*, 18 pp., 2013. doi: 10.14279/tuj.eceasst.57.875

[JR16] M. Johnson, R. Rosebrugh. Unifying set-based, delta-based and edit-based lenses. In A. Anjorin, J. Gibbons, eds., *Proc. of 5th Int. Wksh. on Bidirectional Transformations, BX 2016 (Eindhoven, April 2016)*, v. 1571 of *CEUR Wksh. Proc.*, pp. 1–13, 2016. `http://ceur-ws.org/Vol-1571/paper_13.pdf`

[JRW10] M. Johnson, R. Rosebrugh, R. J. Wood. Algebras and update strategies. *J. of Univ. Comput. Sci.*, v. 16, n. 5, pp. 729–748. doi: 10.3217/jucs-016-05-0729

[JRW12] M. Johnson, R. Rosebrugh, R. J. Wood. Lenses, fibrations and universal translation. *Math. Struct. in Comput. Sci.*, v. 22, n. 1, pp. 25–42, 2012. doi: 10.1017/s0960129511000442

[McK16] J. McKinna. Bidirectional transformations are proof-relevant bisimulations. Extended abstract presented at 2016 ACM SIGPLAN Wksh. on Type-Driven Development, TyDe '16 (Nara, Japan 2016).

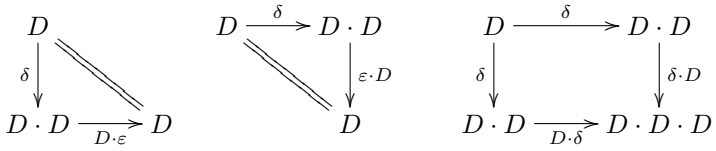
[OCo11] R. O'Connor. Functor is to lens as applicative is to biplate: introducing multiplate. arXiv preprint 1103.2841, 2011. (Paper presented at 2011 ACM SIGPLAN Wksh. on Generic Programming, WGP '11, Tokyo, Sept. 2011.) `https://arxiv.org/abs/1103.2841`

[PS04] J. Power, O. Shkaravska. From comodels to coalgebras: state and arrays. In J. Adámek, S. Milius, eds., *Proc. of 7th Int. Wksh. on Coalgebraic Methods in Computer Science, CMCS '04 (Barcelona, March 2004)*, v. 106 of *Electron. Notes in Theor. Comput. Sci.*, pp. 297–314. Elsevier, 2004. doi: 10.1016/j.entcs.2004.02.041

A Comonads, coalgebras of comonads, comonad morphisms

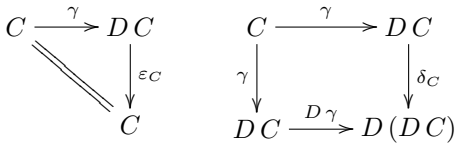
For reference only, we recapitulate the definitions of comonads, coalgebras of comonads, comonad morphisms.

A *comonad* on a category $\mathbb{C}$ is given by a functor $D : \mathbb{C} \to \mathbb{C}$ together with natural transformations $\varepsilon : D \to \mathsf{Id}$ and $\delta : D \to C \cdot C$ such that the diagrams



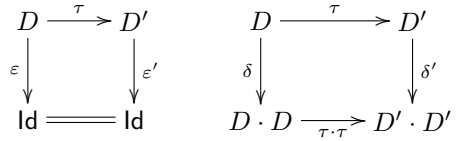
commute.

A *coalgebra* of a comonad (D, ε, δ) is a an object C together with a map $\gamma : C \to D\,C$ such that the diagrams



commute.

A morphism between two comonads (D, ε, δ) and $(D', \varepsilon', \delta')$ on the same category $\mathbb{C}$ is a natural transformation $\tau : D \to D'$ such that the diagrams



commute.

B Coalgebras of a comonad vs morphisms from costate comonads

We prove the following proposition for **Set**, but in fact this proof scales to any monoidal closed category.

Proposition 1 *Given a comonad D, for any set S_0, there is a bijection between coalgebras of D with S_0 as the carrier and morphisms from the costate comonad D^{S_0} to the comonad D.*

Proof. Given a comonad coalgebra structure $\gamma : S_0 \to D\,S_0$, we define a family of maps $\overline{\gamma}$ with components

$$\overline{\gamma}_X : D^{S_0}\,X \to D\,X$$

by

$$\overline{\gamma}_X\,(s_0, v) =_{\mathrm{df}} D\,v\,(\gamma\,s_0)$$

$\overline{\gamma}$ is natural: for $f : X \to Y$, we have

$$\begin{aligned} & D\,f\,(\overline{\gamma}_X\,(s_0, v)) \\ = \;& D\,f\,(D\,v\,(\gamma\,s_0)) \\ = \;& D\,(f \circ v)\,(\gamma\,s_0) \\ = \;& \overline{\gamma}_Y\,(s_0, f \circ v) \\ = \;& \overline{\gamma}_Y\,(D^{S_0}\,f\,(s_0, v)) \end{aligned}$$

$\overline{\gamma}$ is a comonad morphism, because γ satisfies the laws of a comonad coalgebra structure:

$$\begin{array}{l} \varepsilon_X\,(\overline{\gamma}_X\,(s_0, v)) \\ \quad = \quad \varepsilon_X\,(D\,v\,(\gamma\,s_0)) \\ \quad = \quad v\,(\varepsilon_{S_0}\,(\gamma\,s_0)) \\ \quad = \quad v\,s_0 \\ \quad = \quad \varepsilon_X^{S_0}\,(s_0, v) \end{array}$$

$$\begin{array}{l} \delta_X\,(\overline{\gamma}_X\,(s_0, v)) \\ \quad = \quad \delta_X\,(D\,v\,(\gamma\,s_0)) \\ \quad = \quad D\,(D\,v)\,(\delta_{S_0}\,(\gamma\,s_0)) \\ \quad = \quad D\,(D\,v)\,(D\,\gamma\,(\gamma\,s_0)) \\ \quad = \quad D\,(D\,v \circ \gamma)\,(\gamma\,s_0) \\ \quad = \quad D\,(\lambda s_0'.\,\overline{\gamma}_X\,(s_0', v))\,(\gamma\,s_0)) \\ \quad = \quad D\,\overline{\gamma}_X\,(D\,(\lambda s_0'.\,(s_0', v))\,(\gamma\,s_0)) \\ \quad = \quad D\,\overline{\gamma}_X\,(\overline{\gamma}_{D^{S_0}\,X}\,(s_0, \lambda s_0'.\,(s_0', v)) \\ \quad = \quad D\,\overline{\gamma}_X\,(\overline{\gamma}_{D^{S_0}\,X}\,(\delta_X^{S_0}\,(s_0, v))) \end{array}$$

Given a comonad morphism τ, we define a map

$$\underline{\tau} : S_0 \to D\,S_0$$

by

$$\underline{\tau}\,s_0 =_{\mathrm{df}} \tau_{S_0}\,(s_0, \mathsf{id}_{S_0})$$

$\underline{\tau}$ is a comonad coalgebra structure, because τ satisfies the comonad morphism laws:

$$\begin{array}{l} \varepsilon_{S_0}\,(\underline{\tau}\,s_0) \\ \quad = \quad \varepsilon_{S_0}\,(\tau_{S_0}\,(s_0, \mathsf{id}_{S_0})) \\ \quad = \quad \varepsilon_{S_0}^{S_0}\,(s_0, \mathsf{id}_{S_0}) \\ \quad = \quad s_0 \end{array}$$

$$\begin{array}{l} \delta_{S_0}\,(\underline{\tau}\,s_0) \\ \quad = \quad \delta_{S_0}\,(\tau_{S_0}\,(s_0, \mathsf{id}_{S_0})) \\ \quad = \quad D\,\tau_{S_0}\,(\tau_{D^{S_0}\,S_0}\,(\delta_{S_0}^{S_0}\,(s_0, \mathsf{id}_{S_0}))) \\ \quad = \quad D\,\tau_{S_0}\,(\tau_{D^{S_0}\,S_0}\,(s_0, \lambda s_0'.\,(s_0', \mathsf{id}_{S_0}))) \\ \quad = \quad D\,\tau_{S_0}\,(\tau_{D^{S_0}\,S_0}\,(D^{S_0}\,(\lambda s_0'.\,(s_0'\,\mathsf{id}_{S_0}))\,(s_0, \mathsf{id}_{S_0}))) \\ \quad = \quad D\,\tau_{S_0}\,(D\,(\lambda s_0'.\,(s_0'\,\mathsf{id}_{S_0}))\,(\tau_{S_0}\,(s_0, \mathsf{id}_{S_0}))) \\ \quad = \quad D\,(\lambda s_0'.\,\tau_{S_0}\,(s_0', \mathsf{id}_{S_0}))\,(\tau_{S_0}\,(s_0, \mathsf{id}_{S_0})) \\ \quad = \quad D\,\underline{\tau}\,(\tau_{S_0}\,(s_0, \mathsf{id}_{S_0})) \\ \quad = \quad D\,\underline{\tau}\,(\underline{\tau}\,s_0) \end{array}$$

$\overline{(-)}$ is a bijection, as demonstrated by the following calculations.

$$\begin{array}{l} \underline{(\overline{\gamma})}\,s_0 \\ \quad = \quad \overline{\gamma}_{S_0}\,(s_0, \mathsf{id}_{S_0}) \\ \quad = \quad D\,\mathsf{id}_{S_0}\,(\gamma\,s_0) \\ \quad = \quad \gamma\,s_0 \end{array}$$

$$
\begin{aligned}
&\overline{(\underline{\tau})}_X\,(s_0, v)\\
&\quad = \quad D\,v\,(\underline{\tau}\,s_0)\\
&\quad = \quad D\,v\,(\tau_{S_0}\,(s_0, \mathsf{id}_{S_0}))\\
&\quad = \quad \tau_X\,(D^{S_0}\,v\,(s_0, \mathsf{id}_{S_0}))\\
&\quad = \quad \tau_X\,(s_0, v)
\end{aligned}
$$

□

C Cofunctors and opfibrations

We recapitulate the definitions of a cofunctor in the sense of Aguiar [Agu97] and an opfibration (to be precise, an opfibration with chosen lifts, also called an opcleavage) and compare.

A *cofunctor* from a category $\mathbb{D}$ to a category $\mathbb{C}$ is given by an object mapping $F : |\mathbb{C}| \to |\mathbb{D}|$, and an operation taking any object X of $\mathbb{C}$ and any map $f : F\,X \to W$ of $\mathbb{D}$ into a map $f^*_X : X \to Y$ of $\mathbb{C}$ (the *lift* of f) such that $F\,Y = W$. It is required that $(\mathsf{id}_{F\,X})^*_X = \mathsf{id}_X$ and $g^*_Y \circ f^*_X = (g \circ f)^*_X$.

Given a functor $F : \mathbb{C} \to \mathbb{D}$, a map $k : X \to Y$ of $\mathbb{C}$ is said to be *opCartesian* wrt. F, if, for any map $k' : X \to Y'$ of $\mathbb{C}$ and any map $g : F\,Y \to F\,Y'$ of $\mathbb{D}$ such that $F k' = g \circ F\,k$, there exists a unique map $\ell : Y \to Y'$ such that $k' = \ell \circ k$ and $F\,\ell = g$.

An *opcleavage* of a category $\mathbb{C}$ over a category $\mathbb{D}$ is given by a functor $F : \mathbb{C} \to \mathbb{D}$, and an operation taking any object X of $\mathbb{C}$ and any map $f : F\,X \to W$ of $\mathbb{D}$ into an opCartesian map $f^*_X : X \to Y$ of $\mathbb{C}$ (the *lift* of f) such that $F\,f^*_X = f$ (so also $F\,Y = W$). An opcleavage is said to be *splitting*, if additionally $(\mathsf{id}_{F\,X})^*_X = \mathsf{id}_X$ and $g^*_Y \circ f^*_X = (g \circ f)^*_X$.

Notice that in the definition of a cofunctor, F is just an object mapping. In the definition of an opcleavage, F is a functor. Notice also that lifts in an opcleavage are required to be opCartesian.