Electronic Communications of the EASST
Volume 57 (2013)

Proceedings of the
Second International Workshop on
Bidirectional Transformations
(BX 2013)

Delta lenses and opfibrations

Michael Johnson and Robert Rosebrugh

18 pages

Guest Editors: Perdita Stevens, James F. Terwilliger
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Delta lenses and opfibrations

Michael Johnson[1] **and Robert Rosebrugh**[2]

[1] http://www.cs.mq.edu.au/~mike
Department of Computing
Macquarie University, Australia
and
School of Mathematics and Statistics
University of Sydney, Australia

[2] http://www.mta.ca/~rrosebru
Department of Mathematics and Computer Science
Mount Allison University, Canada

Abstract: We compare the delta lenses, also known as d-lenses, of Diskin et al. with the c-lenses, known to be equivalent to opfibrations, already studied by the authors. Contrary to expectation a c-lens is a d-lens but not conversely. This result is surprising because d-lenses appear to provide the same information as c-lenses, and some more besides, suggesting that the implication would be the reverse – a d-lens would appear to be a special kind of c-lens. The source of the surprise can be traced to the way the two concepts deal differently with morphisms in a certain base comma category $(G, 1_{\mathbf{V}})$. Both c-lenses and d-lenses are important because they extend the notion of lens to take account of the information available in known transitions between states and this has important implications in practice.

Keywords: Delta lens, opfibration, view update

1 Introduction

One common asymmetric situation that requires bidirectional programming arises when one has a system and, in some sense or another, a view of that system. The asymmetry arises because the view can always be computed from the system, but in general constructing or reconstructing a system state from a view state may be difficult or even impossible.

An (asymmetric) lens is a framework for attacking this problem. We consider the states of the system, say $\mathbf{S}$, and the states of the view, say $\mathbf{V}$, and the way of computing the view state from the system state is represented by an operation G, often called Get, with $G : \mathbf{S} \longrightarrow \mathbf{V}$. In the reverse direction a lens provides an operation called a Put, P, which yields a system state $\mathbf{S}$, but depends upon knowing both a view state and some other information including the immediately previous system state (the state that we are seeking to update).

Often the system in question is a database, as it will be in many of our examples below, and the view is the result of a query on that database (so the view state is directly computable from the system state). In many applications the view is a major interface to the system, so information is updated by changing the view, and the *view update problem* is the question of how to modify a state of the database to correspond to a given modification of the corresponding view state. Notice that in this situation, as in lenses in general, the Put operation will take as input both the new view state, and further information including the system state immediately before the update, and possibly other information such as the operations that were carried out to achieve the view modification.

Much of the early work on the view update problem, and on lenses in general, concentrated merely on the states, so that $\mathbf{S}$ and $\mathbf{V}$ were simply the *sets* of system states and view states respectively, and the Get and Put could be represented as functions

$$G : \mathbf{S} \longrightarrow \mathbf{V} \quad \text{and} \quad P : \mathbf{V} \times \mathbf{S} \longrightarrow \mathbf{S}.$$

Of course states usually have more structure, typically being represented as directed graphs or categories, with the transitions between states being represented as edges or arrows. Furthermore, in more recent times the present authors and others, especially Diskin et al [4], have argued for the benefits of paying attention to those transitions, and wherever possible, of using them to aid in calculating better solutions to view updating problems.

The transitions between states might be thought of as a record of the changes from one state to another. This is what Diskin et al called the *deltas*. They developed a new notion of lens called a *delta-lens*, which they abbreviate to *d-lens*, and they introduced those lenses in [4] along with very good arguments and practical examples showing why the transitions, the deltas, should be used in calculating Puts.

Independently, both in concept, and in their nomenclature, the present authors and their colleague Wood introduced *c-lenses* [7, 12]. We had been exploring category theoretic approaches to lenses, and had studied lenses in the category **cat** so that the system and view states $\mathbf{S}$ and $\mathbf{V}$ would be categories — the objects of those categories would be the states, and the arrows in those categories would be the transitions between states. However, classical lenses in **cat** don't really take advantage of the transitions in calculating Puts, so we started afresh seeking to develop *category lenses* — those that took advantage of the category theoretic structure of state spaces — and abbreviated the resulting notion to *c-lens*.

The main goal of this paper is to study the relationships between c-lenses and d-lenses. The two concepts are clearly trying to take advantage of the same extra real-world structure. There are superficial differences that can be dealt with fairly easily. But there are also some perplexing differences, chief among them being that c-lenses, like lenses in general, have a put operation that delivers a new system state, while d-lenses are designed to also deliver the transition that shows how the new system state arises from the immediately preceding system state — an apparently significantly stronger requirement. Furthermore, as became apparent through our study, the c-lenses and d-lenses treat morphisms between inputs to the Put operation very differently.

One of the main problems, dealt with in the immediately succeeding section, is to develop a suitable "input category" for P analogous to $\mathbf{V} \times \mathbf{S}$ for classical lenses. It turns out that both c-lenses and d-lenses can be based on variants of a certain comma category $(G, 1_{\mathbf{V}})$ described below. The different treatment of morphisms referred to at the end of the previous paragraph refers to the way c-lenses and d-lenses deal with morphisms of $(G, 1_{\mathbf{V}})$.

Our main result is that, while d-lenses might a priori appear to be stronger than c-lenses, they are in fact more general than c-lenses: If we ignore a connectedness requirement introduced, but not used substantively, in [4], then every c-lens is a d-lens. And the extra generality of d-lenses arises from the looser treatment of morphisms in $(G, 1_{\mathbf{V}})$.

(Alternatively, one could add the connectedness requirement to c-lenses, and then the result still holds: Every c-lens with connected state categories is a d-lens with connected state categories.)

We turn now to the structure of this paper.

First it should be noted that in this introduction we have reviewed G and P, but there are of course axioms, "laws", that they must satisfy and we have for simplicity said nothing about them yet. The PutGet ("Put and then Get") law for example says that if one starts with a view state and some other information, and constructs a system state, then calculating a view state from the resultant system state yields the original view state: A Put should give a system state whose view is the one we Put.

Similarly there are well known GetPut and PutPut laws. A lens which satisfies all three has been called *very well behaved*. In fact, all our lenses, c-lenses and d-lenses, will be very well behaved in the appropriate senses (their own versions of GetPut, PutGet and PutPut will be introduced as they are defined) so an unadorned reference to "lens" should be read as being very well behaved.

For simplicity this paper has been written to deal with updates in the "insert" direction. Some authors treat transitions among states as simply that — a record that one state, the one at the source of the transition, changed to another, the one at the target of the transition. Other authors, especially in the database field, orient their transitions in the direction of increasing information (so that the arrows can be thought of as inclusions of tuples), and "delete" transitions are achieved by traversing an arrow backwards. In most respects the paper can be read with either approach in mind, but for the fullest treatment of c-lenses one should view what we present here as being about inserts, and consider dual approaches (involving $(1_{\mathbf{V}}, G)$) for deletes, and refer to [9] for a treatment of mixed insert and delete updates.

The next section introduces some notation and standard constructions from category theory. The ideas are simple, and fully described, but because they are succinctly described those with limited category theoretic background might want to refer to a standard text (eg [13]), or to skim that section and read on.

The following three sections respectively introduce d-lenses (Section 3) and c-lenses (Section 4) and establish the main results relating them (Section 5). Then in Section 6 we establish some further category theoretic properties of d-lenses.

We end this introduction with a remark about universal properties. The authors have shown that c-lenses satisfy a universal property — the P of a c-lens provides a "least-change update" (in a sense that will be made precise below). Something which is particularly interesting about c-lenses is that this universality is an emergent property — the definition of c-lens merely requires functors $G : \mathbf{S} \longrightarrow \mathbf{V}$ and $P : (G, 1_V) \longrightarrow \mathbf{S}$ satisfying category theoretic expressions of the three standard lens laws for very well behaved lenses. The c-lenses are not required to be least-change lenses — the fact that they are arises from the interaction between the standard laws and the functoriality of the operations.

Interestingly d-lenses, as we will see, satisfy almost exactly the same laws as c-lenses, and the G and P of d-lenses are also functors. Furthermore, the domains and codomains of G and P have the same objects for both kinds of lenses, and mostly the same arrows too. The substantive difference turns out to be that the P of a d-lens need only be defined on some of the arrows of $(G, 1_V)$ (to be precise, arrows for which Gf in the notation of the first display of the next section is an identity arrow).

2 Notation

To describe both d-lenses and c-lenses we need some standard constructions and notation from category theory.

For a functor $G : \mathbf{S} \longrightarrow \mathbf{V}$ denote the "comma category" (with the identity functor) as $(G, 1_{\mathbf{V}})$. The objects of $(G, 1_{\mathbf{V}})$ are pairs $(S, \alpha : GS \longrightarrow V)$ with S and V objects of $\mathbf{S}$ and $\mathbf{V}$ respectively and α an arrow of $\mathbf{V}$. An arrow from $(S, \alpha : GS \longrightarrow V)$ to $(S', \alpha' : GS' \longrightarrow V')$ is a pair of arrows $(f : S \longrightarrow S', g : V \longrightarrow V')$ satisfying $\alpha' Gf = g\alpha$ in $\mathbf{V}$:

$$\begin{tikzcd}
GS \arrow[r, "Gf"] \arrow[d, "\alpha"'] & GS' \arrow[d, "\alpha'"] \\
V \arrow[r, "g"'] & V'
\end{tikzcd}$$

$$(S, \alpha) \xrightarrow[(f,g)]{} (S', \alpha')$$

There is an obvious projection functor from $(G, 1_{\mathbf{V}})$ to $\mathbf{V}$ denoted by $RG : (G, 1_{\mathbf{V}}) \longrightarrow \mathbf{V}$ and defined by $RG(S, \alpha) = V$ and $RG(f, g) = g$.

Since RG is also a functor with codomain $\mathbf{V}$, we can iterate the construction and obtain the comma category $(RG, 1_{\mathbf{V}})$ and denote its projection to $\mathbf{V}$ by $RRG : (RG, 1_{\mathbf{V}}) \longrightarrow \mathbf{V}$. The objects of $(RG, 1_{\mathbf{V}})$ are conveniently described as triples: $(S, \alpha : GS \longrightarrow V, \beta : V \longrightarrow U)$. The arrows from (S, α, β) to $(S', \alpha' : GS' \longrightarrow V', \beta' : V' \longrightarrow U')$, say, are also triples $(g : S \longrightarrow S', h : V \longrightarrow V', k : U \longrightarrow U')$ rendering two squares commutative:

$$\begin{tikzcd}
GS \arrow[r, "Gg"] \arrow[d, "\alpha"'] & GS' \arrow[d, "\alpha'"] \\
V \arrow[r, "h"'] \arrow[d, "\beta"'] & V' \arrow[d, "\beta'"] \\
U \arrow[r, "k"'] & U'
\end{tikzcd}$$

$$(S, \alpha, \beta) \xrightarrow[(g,h,k)]{} (S', \alpha', \beta')$$

We define a functor $\eta_G : \mathbf{S} \longrightarrow (G, 1_{\mathbf{V}})$ on objects of $\mathbf{S}$ by $\eta_G(S) = (S, 1_{GS} : GS \longrightarrow GS)$, and then the extension of η_G to morphisms is obvious. We can also define $\mu_G : (RG, 1_{\mathbf{V}}) \longrightarrow (G, 1_{\mathbf{V}})$. On objects, $\mu_G(S, \alpha, \beta) = (S, \beta\alpha)$. Readers familiar with category theory will rightly suspect that the names of η_G and μ_G suggest they are part of the structure of a monad. Indeed, our c-lenses are precisely algebras for a monad on the category whose objects are functors with codomain $\mathbf{V}$.

The Put functors defined below will be of the form $P : (G, 1_{\mathbf{V}}) \longrightarrow \mathbf{S}$. The PutGet law means that they will be required to satisfy $GP = RG$. For a functor P satisfying that equation, we can define the functor $(P, 1_{\mathbf{V}}) : (RG, 1_{\mathbf{V}}) \longrightarrow (G, 1_{\mathbf{V}})$ on objects by $(P, 1_{\mathbf{V}})(S, \alpha, \beta) = (P(S, \alpha), \beta)$ and note that since $GP = RG$, the domain of β is indeed the object $GP(S, \alpha)$ as required.

3 Delta lenses

The *delta-lens* concept was introduced by Diskin et al in [4] to deal with some of the limitations of the (set-)lenses of Foster et al [5]. Instead of using an unstructured set to model the states of a system (perhaps a database for example), they use the objects of a category to model the states. In [4] no further structure is assumed on the category of models except that, as noted below, they require the category of models to be connected. Using an object of a category to model a state of a database has also been advocated by the authors and R.J. Wood in [10]. This idea is not very new. Indeed, the current authors as well as Diskin and co-authors (for example in [3]) and several others (for example [14]) have argued that the objects of a category of database states should be models of a categorical structure called *sketches*. Sketches have nodes and edges and their models associate sets and functions to them. The idea is similar to entities and entity sets in ERA (Entity-Relationship-Attribute) modelling.

There is an important advantage to having morphisms between states: an update of a specific single state to another state may be represented by a morphism in the category of database states or in the category of view states. This point of view was also adopted by the current authors in [7, 8]. A related way to view updates is as a *process* defined on the states. When the database states and the view states are merely sets, say S and V, then a view update process is a function $u : V \longrightarrow V$ and view updatability requires a lifting to a database state update function (called a *translation*) $t_u : S \longrightarrow S$ [1]. By analogy, when the database states and the view states are categories, say $\mathbf{S}$ and $\mathbf{V}$, then a view update process is a functor $U : \mathbf{V} \longrightarrow \mathbf{V}$ and updatability requires a functor $L_U : \mathbf{S} \longrightarrow \mathbf{S}$. The two points of view can be reconciled as we showed in [12] by specifying a *natural transformation* between the identity functor and the process U. The second point of view is taken in the *edit lenses* [6] of Hofmann et al and in Steven's *monoid of edits* [15], but we will not deal with it further here as these matters are treated with symmetric lenses in another paper currently in preparation.

It is natural to define a database view to be a functor between a category of database states and a category of view states. The functor sends a database state to a view state and each individual database state update to a corresponding update (arrow) between the corresponding view states.

Note that [4] requires the categories of models to have a strongly connected (and of course composable) underlying graph. Their interpretation is that for any database state (object of the model category) there is an update (morphism) to any other state. We do not adopt that restriction in our definition of delta lens since it plays no role.

For a delta lens a *Put* update strategy is defined by a correspondence from view updates to database state updates. The idea is that for any view update (which is specified by a morphism, a *delta*, in the category of view states), there is an update of database states specified by the Put.

The Put for an ordinary (set-)lens specifies the updated database state only as a function of the *initial database state* and the *updated view state*. In the set based paradigm, neither the actual update process for the view update nor the update process for the database states can even be mentioned. Indeed, the only information available about a database or view state is simply the state itself.

Example 1 We briefly consider an example discussed in [4] to illustrate the need for deltas. The database for a firm has one table Person with columns for FirstName, LastName, Department and Birth. The Marketing Department view has a Person table with only the first two attributes.

In this example, the database state has Person rows for Bill Gates, Melinda French and Bill Clinton. The second and third are in the Marketing Department so they appear (without their birthdates) in the view state. What should happen to the database state when in the Marketing Department view Melinda French apparently updates her last name to Gates? Indeed, there are two different updates that could have taken place: Melinda French changed her name as suggested (an update), or Melinda French left the department (a delete) and a new person, Melinda Gates, joined (an insert). In the former case the correct update of the database would result in simply a LastName change. In the latter case the old employee has gone and the database is updated with a new employee, but one whose Birth attribute is unknown.

In order to know which database update should result from the initial database state and the resulting view state (the same in either case), it is clearly important to know which of the two different view state updates occurred. The view state updates with the actual change noted are called *deltas*. In a delta lens the initial database state and the view delta are both needed to obtain the resulting Put value which is a specific database state update.

We should point out that the deltas in [4] are allowed to be view inserts, deletes or updates. As we will mention below, our c-lenses are concerned with formal view inserts. View deletes are handled separately and updates which require a mixture of insert and delete should consider further structure as discussed in more detail in [9].

The coherent specification of a database update (morphism) as the result of a Put on a view update (morphism) requires several equations that are similar to those for an ordinary lens, as well as others (d-PutInc and d-PutId below) that are expected given the categorical structure.

For any category $\mathbf{C}$, we write $|\mathbf{C}|$ for the set (discrete category) of objects of $\mathbf{C}$ and $\mathbf{C^2}$ for the category of arrows of $\mathbf{C}$.

Definition 1 A (very well-behaved) *d-lens* in **cat** is a quadruple $(\mathbf{S},\mathbf{V},G,P)$ where $G:\mathbf{S}\longrightarrow\mathbf{V}$ is a functor and $P:|(G,1_{\mathbf{V}})|\longrightarrow|\mathbf{S^2}|$ is a function and the data satisfy:

(i) d-PutInc: the domain of $P(S,\alpha:GS\longrightarrow V)$ is S

(ii) d-PutId: $P(S,1_{GS}:GS\longrightarrow GS)=1_S$

(iii) d-PutGet: $GP(S,\alpha:GS\longrightarrow V)=\alpha$

(iv) d-PutPut:

$$P(S,\beta\alpha:GS\longrightarrow V\longrightarrow V')=P(S',\beta:GS'\longrightarrow V')P(S,\alpha:GS\longrightarrow V)$$

where S' is the codomain of $P(S, \alpha : GS \longrightarrow V)$

Notice that equation (ii) d-PutId can also be written $P\eta_G(S) = 1_S$ using the η_G defined above. Similarly equation (iv) d-PutPut can also be written $P(\mu_G(S, \alpha, \beta)) = P(S', \beta)P(S, \alpha)$ with S' as in the Definition.

In [4] the Get, G, for a d-lens is initially just a graph morphism and the equations there called (**GetId**) and (**GetGet**) guarantee that the Get is a functor as we require. The equations (**PutInc**$_1$) and (**PutInc**$_2$) in [4] require that the domain of the Put, P, be a pair consisting of an object of $\mathbf{S}$ and an arrow of $\mathbf{V}$ whose domain is the Get of the first object. Such a pair is just an object of $(G, 1_{\mathbf{V}})$, and so an element of the domain of P as we require above. Moreover the equations mentioned also imply the equation d-PutInc required above.

Example 2 Consider again the example discussed above with the Marketing Department view. The ambiguity about which database update should result from the update to Melinda French is resolved when the update is seen as a sequence of two view updates: delete Melinda French, then insert Melinda Gates (with unknown birth). There is an unambiguous Put for each of these view updates: the first Put provides a database delete of Melinda French and the second Put provides a database insert of Melinda Gates. We can extend these considerations to define a d-lens structure on the Get for the firm database. The d-PutPut law implies that the Put for the view update from Melinda French to Melinda Gates provides the second of the possible database updates.

The authors of [4] define a category we will call **DLens**. The objects of **DLens** are arbitrary categories (though they are required to be connected in [4]). The arrows are d-lenses. Now suppose $K = (\mathbf{S}, \mathbf{V}, G_K, P_K)$ and $L = (\mathbf{V}, \mathbf{W}, G_L, P_L)$ are d-lenses with the codomain of the Get for K being the domain of the Get for L. Thus they are composable arrows of **DLens**. Their composite is defined by composing the Get functors G_K and G_L and then defining the Put for the composite by using the Puts from the factors. In detail, the composite LK is the 4-tuple $LK = (\mathbf{S}, \mathbf{W}, G_L G_K, P_{LK})$ where $P_{LK} : |(G_L G_K, 1_{\mathbf{V}})| \longrightarrow |\mathbf{S}^2|$ is defined by $P_{LK}(S, \gamma : G_L G_K S \longrightarrow W) = P_K(S, P_L(G_K S, \gamma : G_L G_K S \longrightarrow W))$. Note that d-PutInc for G_L means P_{LK} is well-defined, and together with d-PutInc for G_K we see that P_{LK} satisfies d-PutInc. The other axioms for a d-lens are easily verified for LK. Finally, each identity functor is the Get of an obvious d-lens and the composition of d-lenses just defined is manifestly associative.

4 c-lenses

In this section we describe the context for our c-lenses and then define them.

Considerations similar to those in the previous section led the authors first to generalize lenses so that instead of a set with no additional structure the domain and codomain of the Get is supposed to be an ordered set, or more generally a category. Then it is natural to require that the Get be an order-preserving mapping in the case of ordered sets, or more generally a functor. We began by considering functors that satisfy the ordinary lens equations in [11]. This direct generalization of ordinary lenses has a very strong consequence for the domain of the Get functor. Indeed, the Get functor is required to be a projection from a product of categories. That is, the domain of the Get functor $\mathbf{S}$ must be the categorical product of the category of view states $\mathbf{V}$ with some other category $\mathbf{C}$. This $\mathbf{C}$ can be thought of as the view complement. Moreover the Get functor $G : \mathbf{S} \longrightarrow \mathbf{V} = G : \mathbf{C} \times \mathbf{V} \longrightarrow \mathbf{V}$ must be essentially the projection $\pi : \mathbf{C} \times \mathbf{V} \longrightarrow \mathbf{V}$. This is not too surprising since the same is true of ordinary lenses: the domain of the Get function is a product of sets where one factor is the codomain of the Get and the Get function is the projection.

Like the authors of [4], we were motivated by wanting to record the change from a view state to its update for use as an input to the Put. We were also motivated by wishing to avoid the strong restriction on Get functors to be only projections. Thus, we also require that the domain of Put for (insert) updates should be a comma category.

The equations in the next definition are very similar to those for a lens, but with the domain of Put generalized to the appropriate comma category.

Definition 2 A *c-lens* in **cat** is a quadruple $(\mathbf{S}, \mathbf{V}, G, P)$ where $G : \mathbf{S} \longrightarrow \mathbf{V}$ and $P : (G, 1_{\mathbf{V}}) \longrightarrow \mathbf{S}$ satisfy

i) c-PutGet: $GP = RG$

ii) c-GetPut: $P\eta_G = 1_{\mathbf{S}}$

iii) c-PutPut: $P\mu_G = P(P, 1_{\mathbf{V}})$

The notation using η and μ is precise and incorporates compactly the requirements both on objects and on arrows. But we do want to emphasise that these axioms are simply category theoretic expressions of the standard lens laws, so here is a sentence or two about each.

First c-PutGet is the usual Put and then Get law which says that starting from an update in $\mathbf{V}$, and of course an appropriate object of $\mathbf{S}$, if we Put and then Get we return to the given $\mathbf{V}$ update. The left hand side says Put and then Get, and the right hand side, RG, says project off the “appropriate object of $\mathbf{S}$” to give simply the original $\mathbf{V}$ update (remembering that R is an endofunctor on $\mathbf{cat}/\mathbf{V}$ which takes G, an object of $\mathbf{cat}/\mathbf{V}$, to another object of $\mathbf{cat}/\mathbf{V}$, which by the defintion of R in Section 2 is the projection from $(G, 1_{\mathbf{V}})$ to $\mathbf{V}$).

Next c-GetPut is the usual Get and then Put law which says that starting from a state S in $\mathbf{S}$ if we apply G to get a state in $\mathbf{V}$ and we do nothing to it (update it with the identity), then applying P returns us to the original state in $\mathbf{S}$. The η appearing in the equation merely takes GS and turns it into the “do nothing” object $(S, 1_{GS})$ of $(G, 1_{\mathbf{V}})$ to which P can then be applied.

Finally, c-PutPut is the usual PutPut law: If we start with an object S of $\mathbf{S}$ and a composable pair of arrows in $\mathbf{V}$ starting at GS then we can either compose the arrows (which is what μ does) and then apply P to the single resulting arrow (the left hand side), or we can successively apply P to the first arrow (while remembering the second) and then apply P to the second arrow (using in $(G, 1_{\mathbf{V}})$ the object S' which resulted from the first application of P).

Notice also how these equations are very similar to the equations for a d-lens. The Put for a c-lens has the category $(G, 1_{\mathbf{V}})$ as its domain, so it is automatically defined on arrows of $(G, 1_{\mathbf{V}})$, but c-GetPut is similar to d-PutId, c-PutGet is similar to d-PutGet and the PutPut laws are also related.

As the authors showed in [12], these equations turn out to be equivalent to requiring that the Get functor G of a c-lens is an instance of a well-known categorical concept: the (split) opfibration where the Put functor assigns the (codomains of the) *opcartesian arrows* in that structure. We briefly describe opfibrations and their interpretation for view updating.

Let $G : \mathbf{S} \longrightarrow \mathbf{V}$ be a functor. The functor G is an *opfibration* if it satisfies the following: for any morphism $\alpha : GS \longrightarrow V$ in $\mathbf{V}$ (that is, an object of $(G, 1_{\mathbf{V}})$) there is a morphism $\hat{\alpha} : S \longrightarrow S'$ such that $G\hat{\alpha} = \alpha$ and furthermore, for any $\gamma : S \longrightarrow S''$, if $G\gamma$ factors as $G\gamma = \alpha'\alpha$ then there is a unique morphism $\gamma' : S' \longrightarrow S''$ satisfying $G\gamma' = \alpha'$ and $\gamma'\hat{\alpha} = \gamma$. The situation is depicted as follows, where the dashed vertical lines indicate the effect of G:

$$\begin{tikzcd}
S \arrow[r, "\hat{\alpha}"] \arrow[rr, bend left, "\gamma"] \arrow[d, dash, dashed] & S' \arrow[r, dotted, "\gamma'"] \arrow[d, dash, dashed] & S'' \arrow[d, dash, dashed] \\
GS \arrow[r, "\alpha"'] & V \arrow[r, "\alpha'"'] & V'
\end{tikzcd}$$

The idea is that the *opcartesian arrow* $\hat{\alpha}$ is a lift of $\alpha : GS \longrightarrow V$. The domain of a c-lens Put is similar to that of a d-lens Put. The resulting opcartesian arrow is a database update whose Get is α. We note that we interpret $\alpha : GS \longrightarrow V$ as a formal insert from GS to V in the category $\mathbf{V}$ of view states. A formal *delete* should be an arrow of $\mathbf{V}$ ending at GS. Thus it should be of the form $\beta : V \longrightarrow GS$. Providing a lift to $\mathbf{S}$ of such arrows is called the *cartesian* property and a functor $G : \mathbf{S} \longrightarrow \mathbf{V}$ that has a cartesian arrow for every $\beta : V \longrightarrow GS$ is called a *fibration*.

The factorization (or *universal*) property required above makes $\hat{\alpha}$ the *best lift* of α. This is a stronger property than is required of a d-lens Put. The reader will already have guessed that the Put for a c-lens structure on G has the codomain S' of the opcartesian arrow as its $P(S, \alpha)$. Conversely given a c-lens structure, the fact that P is a functor (along with the c-PutPut law) allows the definition of an opcartesian arrow for every $\alpha : GS \longrightarrow V$ as shown in the following section.

Example 3 There is a c-lens associated with the Marketing Department view.

Assume that the morphisms of database states are simply inclusions among tables with the following exception: there is a value `unknown` for the Birth attribute and we decree that for a Person database table, a `(F,L,D,unknown)` row is considered to be included in a `(F,L,D,B)` row in any other Person database table for any `B`. That allows any view *insertion* update to have a unique best lifted database update whose result is the Put of the view update. The structure that results is a c-lens which is the d-lens described in Example 2.

5 c-lenses are d-lenses

Let $L = (\mathbf{S}, \mathbf{V}, G, P)$ be a c-lens. We will define a d-lens $L_d = (\mathbf{S}, \mathbf{V}, G_d, P_d)$. For G_d we just take G. The data for L define a put functor P that determines an object $P(S, \alpha)$ of $\mathbf{S}$ for each object (S, α) of $(G, 1_{\mathbf{V}})$. To define a P_d we need to define a function on objects of $(G, 1_{\mathbf{V}})$ that returns an arrow of $\mathbf{S}$. We use functoriality of P which includes that P is defined on arrows of $(G, 1_{\mathbf{V}})$. Let (S, α) be an object of $(G, 1_{\mathbf{V}})$. We need to define $P_d(S, \alpha)$ to be an arrow of $\mathbf{S}$. Now there is an arrow $(1_S,\alpha): 1_{GS} \longrightarrow \alpha$ in $(G,1_{\mathbf{V}})$:

$$\begin{tikzcd}
GS \arrow[r, "G1_S"] \arrow[d, "1_{GS}"'] & GS \arrow[d, "\alpha"] \\
GS \arrow[r, "\alpha"'] & V
\end{tikzcd}$$

$$(S,1_S) \xrightarrow[(1_S,\alpha)]{} (S,\alpha)$$

Define $P_d(S,\alpha)$ to be the arrow $P(1_S,\alpha): S \longrightarrow S'$ with codomain the object $S' = P(S,\alpha)$. Note that equation c-GetPut on objects guarantees that the domain of $P_d(S,\alpha)$ is S.

Proposition 1 *Let $L = (\mathbf{S},\mathbf{V},G,P)$ be a c-lens and $L_d = (\mathbf{S},\mathbf{V},G_d,P_d)$ with G_d and P_d as just defined. Then L_d is a d-lens.*

Proof. The required data are already in place, so we have four equations to check.

(i) d-PutInc: the domain of $P_d(S,\alpha : GS \longrightarrow V)$ is S. We already observed this.

(ii) d-PutId: $P_d(S,1_{GS} : GS \longrightarrow GS) = 1_S$. By definition, $P_d(S,1_{GS}) = P(1_S,1_{GS})$ and the morphism $(1_S,1_{GS}) : 1_{GS} \longrightarrow 1_{GS}$ is an identity in $(G,1_{\mathbf{V}})$ and since P is a functor, its value after application of P is 1_S as required.

(iii) d-PutGet: $GP_d(S,\alpha : GS \longrightarrow V) = \alpha$. By definition $GP_d(S,\alpha) = GP(1_S,\alpha)$ and by the c-PutGet law, $GP(1_S,\alpha) = RG(1_S,\alpha) = \alpha$ as required.

(iv) d-PutPut: $P_d(S,\beta\alpha : GS \longrightarrow V \longrightarrow V') = P_d(S',\beta : GS' \longrightarrow V')P_d(S,\alpha : GS \longrightarrow V)$ with S' the codomain of $P_d(S,\alpha)$. This eventually follows from the c-lens law c-PutPut. We make a couple of preliminary observations. First, note that $P_d(S,\alpha) = P(1_S,\alpha)$ is the opcartesian morphism denoted $\hat{\alpha}$ above, so $G\hat{\alpha} = \alpha$. However, it is *not* the case that $\widehat{Gc} = c$ for an arbitrary morphism $c : S \longrightarrow S'$ of $\mathbf{S}$: not every morphism in $\mathbf{S}$ need be the opcartesian morphism over its image under G. Next, we remind the reader that the c-GetPut equation says that applying P to η_G of a morphism c in $\mathbf{S}$ returns c. Finally, the morphism of $(G,1_{\mathbf{V}})$:

$$\begin{tikzcd}
GS \arrow[r, "Gc"] \arrow[d, "Gc"'] & GS' \arrow[d, "1_{GS'}"] \\
GS' \arrow[r, "1_{GS'}"'] & GS'
\end{tikzcd}$$

$$GS \xrightarrow[(Gc,1_{GS'})]{} GS'$$

is sent by P to the unique (fill-in) morphism over $1_{GS'}$ between the opcartesian morphism $\widehat{Gc}$ and c. And again, remember that $\widehat{Gc}$ is not in general equal to c.

Next consider the domain of an instance of the c-PutPut equation that we need. It is a morphism in the iterated comma category:

$$\begin{tikzcd}
GS \arrow[r, "G(1_S)"] \arrow[d, "1_{GS}"'] & GS \arrow[d, "\alpha"] \\
GS \arrow[r, "\alpha"'] \arrow[d, "1_{GS}"'] & V \arrow[d, "\beta"] \\
GS \arrow[r, "\beta\alpha"'] & V'
\end{tikzcd}$$

$$(S, 1_{GS}, 1_{GS}) \xrightarrow[(1_{GS},\alpha,\beta)]{} (S, \alpha, \beta)$$

The left hand side of c-PutPut first applies μ_G to obtain:

$$\begin{tikzcd}
GS \arrow[r, "G(1_S)"] \arrow[d, "1_{GS}"'] & GS \arrow[d, "\beta\alpha"] \\
GS \arrow[r, "\beta\alpha"'] & V'
\end{tikzcd}$$

$$(S, 1_{GS}) \xrightarrow[(1_{GS},\beta\alpha)]{} (S', \beta\alpha)$$

Applying P to this square gives the opcartesian morphism for $\beta\alpha$, namely $\widehat{\beta\alpha} : S \longrightarrow S''$. This morphism is also the left hand side of the d-PutPut equation.

The right hand side of c-PutPut first applies $(P, 1_{\mathbf{V}})$ to the two vertically stacked squares above to obtain the $(G, 1_{\mathbf{V}})$ morphism:

$$\begin{tikzcd}
GS \arrow[r, "G(\hat{\alpha})"] \arrow[d, "1_{GS}"'] & GS' \arrow[d, "\beta"] \\
GS \arrow[r, "\beta\alpha"'] & V'
\end{tikzcd}$$

$$(S, 1_{GS}) \xrightarrow[(\hat{\alpha},\beta\alpha)]{} (S', \beta)$$

Applying P to this also gives $\widehat{\beta\alpha} : S \longrightarrow S''$ by c-PutPut. To finish we need to analyze this morphism $(\hat{\alpha}, \beta\alpha)$ further. It can be factorized in $(G, 1_{\mathbf{V}})$ as follows:

$$\begin{tikzcd}
GS \arrow[r, "G(1_S)"] \arrow[d, "1_{GS}"'] & GS \arrow[r, "G(\hat{\alpha})"] \arrow[d, "\alpha"] & GS' \arrow[r, "G(1_{S'})"] \arrow[d, "1_{GS'}"] & GS' \arrow[d, "\beta"] \\
GS \arrow[r, "\alpha"'] & V \arrow[r, "1_V"'] & GS' \arrow[r, "\beta"'] & V'
\end{tikzcd}$$

$$(S, 1_{GS}) \xrightarrow[(1_S,\alpha)]{} (S, \alpha) \xrightarrow[(\hat{\alpha},1_V)]{} (S', 1_{GS'}) \xrightarrow[(1_{S'},\beta)]{} (S', \beta)$$

Now recognize that applying P to the first and last arrows gives $\hat{\alpha} : S \longrightarrow S'$ and $\hat{\beta} : S' \longrightarrow S''$ as noted above. Recall that $\alpha = G(\hat{\alpha})$ and so the middle arrow of the three is, as noted above, the unique fill-in morphism between $\hat{\alpha} = \widehat{G(\hat{\alpha})}$ and $\hat{\alpha}$ over the identity on S', so it must be the identity on S'. Hence applying P to the composite of all three gives $\hat{\beta}\hat{\alpha}$ since the middle (identity) arrow goes to the identity. But we recognize $\hat{\beta}\hat{\alpha}$ as $P_d(S', \beta : GS' \longrightarrow V')P_d(S, \alpha : GS \longrightarrow V)$, the right hand side of d-PutPut, and so the proof is complete. □

It is worth noting that what we have shown in part (iv) is essentially that opcartesian morphisms compose. This fact is not new, but we feel that the proof above provides insight into the relation between c-lenses and d-lenses. In particular, among the three squares above, the middle square is a morphism that cannot be lifted by a d-lens, while the functoriality of the Put for a c-lens is what we used in the proof. This different treatment of morphisms of $(G, 1_V)$ is at the heart of the relationship.

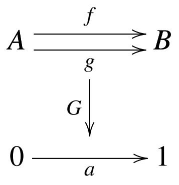
If two c-lenses have composable Get functors, then the c-lenses themselves can be composed: The construction parallels the composition of d-lenses described above in defining **DLens**. Furthermore each identity functor is the Get of a trivial c-lens. Thus there is a category that we call **CLens** with categories as objects and c-lenses as arrows. Consequently:

Proposition 2 *__CLens__ is a subcategory of __DLens__.*

The following example shows that not every d-lens is a c-lens, and hence that **CLens** is a not a full subcategory of **DLens**. In other words, the subcategory inclusion just noted is strict — this is expressed by not being full since the objects of interest, the lenses, are morphisms.

Example 4 Let **S** be the category with two objects A and B and exactly two (different!) non-identity arrows f and g, both from A to B. Let **V** be the category with two objects 0 and 1 and exactly one non-identity arrow a from 0 to 1. The functor G sends A to 0 and B to 1, so both f and g go to a as in the diagram below.

$$\begin{tikzcd} A \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & B \\ & \\ 0 \arrow[r, "a"'] & 1 \end{tikzcd}$$



Now G is *not* an opfibration, so it is not a c-lens. Indeed, one of f or g must be the non-trivial opcartesian arrow and then there is no factorization through a unique fill-in for the other. On the other hand, if we take G as Get and define the Put by $P(A, GA = 0 \xrightarrow{a} 1) = f$ (or g for that matter) we obtain a perfectly good d-lens. The only other objects of $(G, 1_{\mathbf{V}})$, the domain of P, involve identity arrows, and the equations are clearly satisfied.

We would argue that the example shows that the d-lens notion is slightly more general than is desirable (we say "slightly" because the difference is just whether or not P is defined on some of the arrows of $(G, 1_{\mathbf{V}})$). If **S** has two parallel "deltas", it may not be possible to determine a canonical Put. Of course our example is artificially minimal, but it is not artificial. It can arise anytime that there is a state-space with parallel deltas and that can easily happen in categories of bags or strings for example. We believe that the opfibration requirements for a c-lens are exactly what one would want to ensure canonical (least-change) updates, but of course there is room for more empirical study to determine when the extra generality of d-lenses might be useful.

6 More on delta lenses

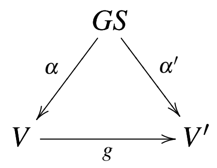
We noted in the last section that the different treatment of morphisms of $(G, 1_{\mathbf{V}})$ by c-lenses and d-lenses results in c-lenses satisfying a universal property, while d-lenses need not. Indeed, it has even been said that d-lenses "are not functorial" — after all, the P for a d-lens is defined on the discrete category, the set, $|(G, 1_{\mathbf{V}})|$, so it "ignores the arrows" of $(G, 1_{\mathbf{V}})$. In fact, d-PutPut ensures that the P of a d-lens extends uniquely to some of the arrows of $(G, 1_{\mathbf{V}})$ — those arrows of the form $(1_S, \alpha)$ as in the leftmost square of the three square display near the end of the proof of Proposition 1. Such squares might be called triangles since the top of the square is degenerate (being the identity morphism $G1_S$). The put of a d-lens is in fact functorial on those triangles.

In this section we develop this in detail.

Furthermore d-lenses are not, in contrast to c-lenses, the algebras for a monad R on $\mathbf{cat}/\mathbf{V}$. Even extending P to the above triangles in $(G, 1_{\mathbf{V}})$ we see that the domain of P does not include the morphisms in the image of η_G, so the identity law for the monad cannot be required of P.

Nevertheless, the requirements for a d-lens can be expressed in terms of certain algebras for a *semi-monad* on $\mathbf{cat}/\mathbf{V}$ which we now describe.

We need some notation. In this section we denote the discrete category with the same objects as $\mathbf{S}$ by $\mathbf{S}_0$ and the inclusion functor by $I_{\mathbf{S}} : \mathbf{S}_0 \longrightarrow \mathbf{S}$ (suppressing the subscript when possible). For a functor $G : \mathbf{S} \longrightarrow \mathbf{V}$, we will also denote the composite $GI_{\mathbf{S}}$ by G_0. Next we consider the comma category $(G_0, 1_{\mathbf{V}})$. The objects are the same as those of $(G, 1_{\mathbf{V}})$, namely pairs $(S, \alpha : G_0S \longrightarrow V)$ and remember that $G_0S = GS$. However there are fewer morphisms since the only morphisms of $\mathbf{S}_0$ are identities. A morphism from (S, α) to $(S', \alpha' : GS' \longrightarrow V')$ is a pair $(f : S \longrightarrow S', g : V \longrightarrow V')$ satisfying $\alpha' Gf = g\alpha$ as before, but since f must be an identity arrow, so is Gf and the equation reduces to $\alpha' = g\alpha$. We might as well think of morphisms as triangles:


$$\begin{tikzcd} & GS \arrow[dl, "\alpha"'] \arrow[dr, "\alpha'"] & \\ V \arrow[rr, "g"'] & & V' \end{tikzcd}$$

As before, there is a projection functor we denote $R_0G : (G_0, 1_{\mathbf{V}}) \longrightarrow \mathbf{V}$ which remembers the codomains. Our construction has defined a functor

$$R_0 : \mathbf{cat}/\mathbf{V} \longrightarrow \mathbf{cat}/\mathbf{V}$$

We have not specified the action of R_0 on morphisms of $\mathbf{cat}/\mathbf{V}$ but the reader can easily fill in the details.

Moreover, since R_0G has codomain $\mathbf{V}$, we can iterate the construction and obtain $R_0R_0G : ((R_0G)_0, 1_{\mathbf{V}}) \longrightarrow \mathbf{V}$. It is worth explicitly describing the domain of this functor. The objects are pairs consisting of an object $(S, \alpha : G_0S \longrightarrow V)$ of $(G_0, 1_{\mathbf{V}})$ and a morphism of $\mathbf{V}$ from $V = R_0G(S, \alpha)$, say, $\beta : V \longrightarrow U$. We write this object as (S, α, β). A morphism is again a "triangle", so is specified by a $g : U \longrightarrow U'$ from (S, α, β) to (S, α, β') satisfying $g\beta = \beta'$:

$$\begin{tikzcd}
 & G_0S \arrow[d, "\alpha"'] & \\
 & V \arrow[dl, "\beta"'] \arrow[dr, "\beta'"] & \\
U \arrow[rr, "g"'] & & U'
\end{tikzcd}$$

With the same formulas as before, we can define functors $\eta_{G_0} : \mathbf{S}_0 \longrightarrow (G_0, 1_{\mathbf{V}})$ and $\mu_{G_0} : ((R_0G)_0, 1_{\mathbf{V}}) \longrightarrow (G_0, 1_{\mathbf{V}})$. Moreover, it is easy to see that $R_0G\mu_{G_0} = R_0R_0G$. We also have $R_0G\eta_{G_0} = G_0$. Thus μ_{G_0} defines a morphism of $\mathbf{cat}/\mathbf{V}$ from R_0R_0G to R_0G and η_{G_0} defines a morphism from G_0 to R_0G.

Now μ_{G_0} is again the component at G of a natural transformation we call μ^0 from R_0R_0 to R_0. However, η_{G_0} is now the G component of a natural transformation η^0 whose domain is *not* the identity on $\mathbf{cat}/\mathbf{V}$, but rather is the functor sending G to G_0. The codomain of the natural transformation is, of course, still R_0. Nevertheless, R_0 and μ^0 do provide the structure of a *semi-monad* on $\mathbf{cat}/\mathbf{V}$. The requirement for this is only that the following associative law holds:

$$\begin{tikzcd}
R_0R_0R_0 \arrow[r, "R_0\mu^0"] \arrow[d, "\mu^0R_0"'] & R_0R_0 \arrow[d, "\mu^0"] \\
R_0R_0 \arrow[r, "\mu^0"'] & R_0
\end{tikzcd}$$

This is easy but tedious to verify.

An algebra for this semi-monad is a pair consisting of an object G of $\mathbf{cat}/\mathbf{V}$ and a morphism P in $\mathbf{cat}/\mathbf{V}$ from R_0G to G satisfying the equation:

$$\begin{tikzcd}
R_0R_0G \arrow[r, "\mu^0G"] \arrow[d, "R_0P"'] & R_0G \arrow[d, "P"] \\
R_0G \arrow[r, "P"'] & G
\end{tikzcd}$$

Notice that the required $P : R_0G \longrightarrow G$ is determined by a functor $P_0 : (G_0, 1_{\mathbf{V}}) \longrightarrow \mathbf{S}$ satisfying $GP_0 = R_0G$ and we will denote semi-monad algebras (G, P_0).

Proposition 3 *A d-lens $(\mathbf{S}, \mathbf{V}, G, P)$ determines an algebra (G, P_0) for the semi-monad (R_0, μ^0) satisfying $P_0\eta^0G = P_0\eta_{G_0} = I_{\mathbf{S}}$, and conversely.*

Proof. Let $(\mathbf{S}, \mathbf{V}, G, P)$ be a d-lens. We have the data for an algebra once we extend P to a functor $P_0 : (G_0, 1_{\mathbf{V}}) \longrightarrow \mathbf{S}$ satisfying $GP_0 = R_0G$. On objects we define $P_0(S, \alpha) = d_1(P(\alpha))$ where d_1 is codomain. Now suppose that $\beta\alpha : GS \longrightarrow V \longrightarrow V' = \alpha' : GS \longrightarrow V$ so that β is a morphism of $(G_0, 1_{\mathbf{V}})$ from α to α'. Define $P_0(\beta) = P(S', \beta)$ where S' is the codomain of $P(S, \alpha)$. It is precisely d-PutPut which ensures that $P_0(\beta)$ is well-defined. It is easy to se that P_0 respects identities and composition in $(G_0, 1_{\mathbf{V}})$. Now on objects, $R_0G(S, \alpha : G_0S \longrightarrow V) = V$ and $GP_0(S, \alpha) = Gd_1(P(\alpha))$, but by d-PutGet, $GP(\alpha) = \alpha$, whose codomain is V. On an arrow β as above, $GP_0(\beta) = GP(\beta) = \beta = R_0G(\beta)$ again using d-PutGet.

For the algebra equation we have only to show that the following square commutes:

$$\begin{tikzcd}
((R_0G)_0, 1_{\mathbf{V}}) \arrow[r, "\mu_{G_0}"] \arrow[d, "(P_0, 1_{\mathbf{V}})"'] & (G_0, 1_{\mathbf{V}}) \arrow[d, "P_0"] \\
(G_0, 1_{\mathbf{V}}) \arrow[r, "P_0"'] & \mathbf{S}
\end{tikzcd}$$

On objects this requires that $P_0(\mu_{G_0}(S, \alpha, \beta)) = P_0((P_0, 1_{\mathbf{V}})(S, \alpha, \beta))$. The left hand side is $P_0(S, \beta\alpha) = d_1(P(\beta\alpha)$. The right hand side evaluates as $P_0(d_1(P(\alpha), \beta)) = d_1(P(\beta))$, but by d-PutPut, $d_1(P(\beta\alpha)) = d_1(P(\beta))$. The argument for morphisms is similar.

Furthermore, for any object S of $\mathbf{S}_0$, we have $P_0(\eta_{G_0}(S)) = P_0(S, 1_{GS}) = d_1P(S, 1_{GS})$ which is S by d-PutId.

Next we show that a semi-monad algebra (G, P_0) satisfying the extra equation determines a d-lens. Of course, $\mathbf{S}$, $\mathbf{V}$ and G are as for the algebra. We need to define $P : |(G, 1_{\mathbf{V}})| \longrightarrow |\mathbf{S}^2|$. Suppose that $(S, \alpha : GS \longrightarrow V)$ is an object of $(G, 1_{\mathbf{V}})$. Then there is a morphism $\alpha : (S, 1_{GS}) \longrightarrow (S, \alpha)$ in $(G_0, 1_{\mathbf{V}})$ and we define $P(S, \alpha) = P_0(\alpha)$, which is a morphism of $\mathbf{S}$.

We need to check the conditions for a d-lens. It is our extra equation which guarantees d-PutInc since it says $P_0(S, 1_{GS}) = S$, and $P_0(S, 1_{GS})$ is the domain of $P(S, \alpha)$. Since P_0 is a functor, $1_S = P_0(1_{GS}) : P_0(S, 1_{GS}) \longrightarrow P_0(S, 1_{GS})$ which proves d-PutId. To see that $GP(S, \alpha) = \alpha$ (d-PutGet) just notice that $GP_0 = R_0G$ implies $GP(S, \alpha) = GP_0(\alpha) = \alpha$ (where the second α is the morphism of $(G_0, 1_{\mathbf{V}})$).

Finally, for d-PutPut, we have to show that

$$P(S, \beta\alpha : GS \longrightarrow V \longrightarrow V') = P(S', \beta : GS' \longrightarrow V')P(S, \alpha : GS \longrightarrow V)$$

where S' is the codomain of $P(S, \alpha : GS \longrightarrow V)$. Now $P(S, \beta\alpha : GS \longrightarrow V \longrightarrow V') = P_0(\beta\alpha) = P_0(\beta)P_0(\alpha)$ by definition and functoriality of P_0 where $\alpha : (S, 1_{GS}) \longrightarrow (S, \alpha)$ and $\beta : \alpha \longrightarrow \beta\alpha$ are in $(G_0, 1_{\mathbf{V}})$. Thus the second factor is $P(S, \alpha)$ by definition. In the first factor we need to show that $P_0(\beta : \alpha \longrightarrow \beta\alpha) = P_0(\beta : 1_{GS'} \longrightarrow \beta)$. To see this we consider the semi-monad algebra equation for (G, P_0) applied to the arrow $\beta : (S, \alpha, 1_{GS'}) \longrightarrow (S, \alpha, \beta)$:

$$\begin{tikzcd}
& G_0S \arrow[d, "\alpha"'] & \\
& V \arrow[dl, "1_{GS'}"'] \arrow[dr, "\beta"] & \\
V' \arrow[rr, "\beta"'] & & V'
\end{tikzcd}$$

Recall that S' is $P_0(\alpha : GS \longrightarrow V)$. The image of that arrow under $\mu_{G_0} P_0$ is clearly $P_0(\beta : \alpha \longrightarrow \beta\alpha)$. On the other hand, applying $P_0(P_0, 1_{\mathbf{V}})$ to $\beta : (S, \alpha, 1_{GS'}) \longrightarrow (S, \alpha, \beta)$, we first obtain

$$\begin{tikzcd} & S' \arrow[dl, "1_{GS'}"'] \arrow[dr, "\beta"] & \\ V' \arrow[rr, "\beta"'] & & V' \end{tikzcd}$$

again since $S' = P_0(\alpha : GS \longrightarrow V)$, and applying P_0 to the displayed morphism β gives $P_0(\beta : 1_{GS'} \longrightarrow \beta)$. The semi-monad algebra equation tells us that the two results are equal and that completes the proof. □

Thus we have shown that a d-lens has a P which extends to P_0 which is functorial on the subcategory $(G_0, 1_{\mathbf{V}})$ of $(G, 1_{\mathbf{V}})$ determined by the "triangular" morphisms. Furthermore (G, P_0) is an algebra for the semi-monad R_0, and the extra equation in Proposition 3 is analogous to the identity axiom for an algebra, but takes into account the restriction that we have here — with only triangular morphisms we can only require the identity axiom to work on S_0, the objects of S, and not on the arrows of S whose images under η_G are not triangular.

7 Conclusions

The two notions of c-lens and d-lens are both designed to advance the theory of lenses, and their applicability in real-world update problems, by incorporating the available information about transitions. If one has just modified a view, and one wants to update the entire system to take account of the view modification, it is very valuable to be able to take account of the nature of the modification that was made (rather than just the resulting final view state).

Thus, it is important to understand the two notions and to develop a precise treatment of their interrelationships.

This paper has shown that they are related in quite the opposite way to what might be expected on examining them initially — c-lenses are special cases of d-lenses and not vice-versa. Furthermore it has determined what is behind that unexpected finding: The different treatments of morphisms of $(G, 1_{\mathbf{V}})$ (a matter that would have remained invisible but for a detailed category theoretic analysis).

In many respects, the two notions, while distinct, do have more in common than their definitions might suggest. While c-lens Puts return objects rather than arrows, they do determine uniquely an arrow of **S** analogous to d-lens Puts. While d-lens Puts are only defined on objects of $(G, 1_{\mathbf{V}})$, they can be extended to be defined and functorial on the triangular arrows of $(G, 1_{\mathbf{V}})$ (of course c-lens Puts are defined and functorial on all of the objects and arrows of $(G, 1_{\mathbf{V}})$). While c-lenses are the algebras for the monad R, d-lenses cannot be, but they are the algebras for a closely related semimonad R_0 which satisfy an axiom analogous to the identity axiom for monad algebras. Nevertheless, their differences are still important.

Practical implications and examples will be taken up elsewhere, but in summary the differences in practice are that d-lenses are more general, permitting as they do update strategies that are coherent in the sense that they satisfy d-PutPut, but not necessarily canonical, while c-lenses are more specific providing as they do universal solutions to view update problems.

The precise benefits of one over the other in practice await detailed empirical study.

Acknowledgements: The authors gratefully acknowledge the support of the Australian Research Council and the Natural Sciences and Engineering Research Council of Canada.

Bibliography

[1] Bancilhon, F. and Spyratos, N. (1981) Update semantics of relational views, *ACM Transactions on Database Systems* **6**, 557–575.

[2] Bohannon, A., Vaughan, J. and Pierce, B. (2006) Relational Lenses: A language for updatable views. *Proceedings of Principles of Database Systems (PODS)*, 338–347. doi:10.1145/1142351.1142399

[3] Diskin, Z. and Cadish, B. (1995) Algebraic graph-based approach to management of multidatabase systems. In *Proceedings of The Second International Workshop on Next Generation Information Technologies and Systems (NGITS '95).*

[4] Diskin, Z., Yingfei Xiong and Czarnecki, K. (2011) From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case, *Journal of Object Technology* **10**, 6:1–25. doi:10.5381/jot.2011.10.1.a6

[5] Foster, J., Greenwald, M., Moore, J., Pierce, B. and Schmitt, A. (2007) Combinators for bi-directional tree transformations: A linguistic approach to the view update problem. *ACM Transactions on Programming Languages and Systems* **29**. doi:10.1145/1232420.1232424

[6] Hofmann, M., Pierce, B. and Wagner D. (2012) Edit lenses. In ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), 495–508.

[7] Johnson, M. and Rosebrugh, R. (2001) View updatability based on the models of a formal specification. In Proceedings of Formal Methods Europe, *Lecture Notes in Computer Science* **2021**, 534–549.

[8] Johnson, M. and Rosebrugh, R. (2007) Fibrations and universal view updatability. *Theoretical Computer Science* **388**, 109–129. doi:10.1016/j.tcs.2007.06.004

[9] Johnson, M. and R. Rosebrugh, R. (2012) Lens put-put laws: monotonic and mixed. *Electronic Communications of the EASST* **49**, 1–13.

[10] Johnson, M., Rosebrugh, R. and Wood, R. J. (2002) Entity-relationship-attribute designs and sketches. *Theory and Applications of Categories* **10**, 94–112.

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

[12] Johnson, M., Rosebrugh, R. and Wood, R. J. (2012) Lenses, fibrations and universal translations. *Mathematical Structures in Computer Science* **22**, 25–42. doi:10.1017/S0960129511000442

[13] Pierce, B. (1991) *Basic category theory for computer scientists*. MIT Press.

[14] Piessens, F. and Steegmans, E. (1995) Categorical data specifications. *Theory and Applications of Categories* **1**, 156–173.

[15] Stevens, P. (2008) Towards an algebraic theory of bidirectional transformations. In Graph Transformations, *Lecture Notes in Computer Science* **5214**, 1–17.