Under consideration for publication in Math. Struct. in Comp. Science

Lenses, fibrations, and universal translations

Michael Johnson†

School of Mathematics and Computing

Macquarie University

Robert Rosebrugh†

Department of Mathematics and Computer Science

Mount Allison University

RJ Wood†

Department of Mathematics and Statistics

Dalhousie University

Received 2010-08-23. Revised 2011-05-16

This article is extends the "lens" concept for view updating in Computer Science beyond the categories of sets and ordered sets. It is first shown that a constant complement view updating strategy also corresponds to a lens for a categorical database model. A variation on the lens concept called c-lens is introduced, and shown to correspond to the categorical notion of Grothendieck opfibration. This variant guarantees a universal solution to the view update problem for functorial update processes.

1. Introduction

In a modern database system the instantaneous *semantics* (a database state) is usually taken to be a set of "tables", also known as relations. This is based on the the idea that a data object is specified by a record which is a list of field values. Then a table is a set of records and a database state is a set of tables. For example, an address book entry (record) will have appropriate numerical and string fields. In the *syntax* for a database, the address book table signature lists the fields and their types. A contacts list database *schema* might include an address book table signature. A table for the address book signature is a set (not a list!) of such records. The tables may be required to satisfy integrity rules such as external references. For example, an address record could be required to refer to a record, for example a person, in another table.

The specification of table signatures and integrity rules is the purpose of a database definition language (DDL). A *database schema* is naturally defined as a correct instance

† Research partially supported by grants from the Australian Research Council and NSERC Canada.

of some DDL. There is a variety database definition languages in use, but by far the most common is SQL.

For a database schema (for example a set of DDL statements in SQL), the *database states* are the valid ways of populating the database schema, usually the tables. We will have much more to say about the structure of a database schema and database states below.

A basic requirement of database states is that they may be updated. Updates may be additions or deletions of records, or modification of some fields. An *update* u is often described as a process that may be applied to any database state and determines a new database state. Thus an update process is an endomorphism on states.

A *view definition* consists of a database schema derived from another database schema and determines an assignment of database states S to *view states* V. So there is at least a *view definition mapping* from S to V. In SQL views are very limited: a view definition describes only a single derived table, but this restriction can be safely ignored.

Combining these two concepts, a *view update* u is an endomorphism of view states. Then the *view update problem* is as follows:

given a view definition $g : S \longrightarrow V$ and an update $u : V \longrightarrow V$ of the view states, when is there a
compatible update (known as a *translation*) $t_u : S \longrightarrow S$ of the database states?

For t_u to be a compatible update (a translation) means that $gt_u = ug$, that is, the following diagram commutes (as noted by (Bancilhon and Spyratos 1981)):

$$\begin{array}{ccc} S & \overset{t_u}{\dashrightarrow} & S \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle g} \\ V & \underset{u}{\longrightarrow} & V \end{array}$$

Of course this is a lifting problem: for a view update u, we ask when ug can be lifted along g to some t_u. If S and V are sets and g is surjective (surjectivity of g is commonly required), then any section of g would provide a solution. However it is natural also to require the translation of the identity view update to be the identity, and to impose other conditions discussed below so that this obvious suggestion is not adequate. The view updates u under consideration are not necessarily arbitrary, but usually those view updates for which the view update problem is required to have a solution should be composable and include the identity. That is, they are required to be a monoid.

The view update problem has a long history in the database literature dating at least to the 1980's. Perhaps the most influential consideration of the problem is (Bancilhon and Spyratos 1981). Their main result amounts to a requirement that there be a product decomposition of the database states with the view states as one factor and with the second factor called the "complement" view. In this case the view update problem has a simple solution with the translation being constant on the second factor. The resulting view update solution is called the "constant complement" strategy.

Several years ago, B. Pierce and collaborators (Bohannon *et al.* 2006), (Foster *et al.* 2007) introduced a concept they called "lens" for a mapping $g : S \longrightarrow V$. A lens has a "Put" mapping $p : V \times S \longrightarrow S$ that satisfies additional equations. A lens provides solutions to view update problems for a view definition mapping g. The equations are strong enough to require that g be a projection, and they showed that a lens for a view definition mapping corresponds to a translator in the sense of (Bancilhon and Spyratos 1981).

As far as we know, the lens equations were first considered in the early 1980's by F. Oles (Oles 1982), (Oles 1986) in a study of abstract models of storage. Oles (as reported in (O'Hearn and Tennent 1995)) also characterized models of the equations in sets as projections. In the 1990's M. Hoffman and B. Pierce (Hofmann and Pierce 1995) also considered the lens equations in their study of typing for programming languages.

At about the same time as the relationship of lenses to constant complement update strategies was noticed, S. Hegner wrote about "update strategies" for a "closed family of updates" (Hegner 2004). For Hegner, the database states should be treated as an ordered set rather than a discrete abstract set. This makes very good sense: if a database state is a set of tables (relations), then there is an obvious partial order among them given by inclusions. Hegner's definition of update strategy includes being a lens in the sense appropriate to the category of partially ordered sets. As we review below, a lens structure actually determines an update strategy.

The present authors recently showed in (Johnson *et al.* 2010) that the lens equations are equivalent to those satisfied by an algebra for a well-known monad on a slice category of a category with products. This clarifies the constant complement approach, and formally unifies the approaches of Bancilhon and Spyratos and that of Hegner.

Rather than treat the database states as given abstractly, as in (Bancilhon and Spyratos 1981) or in (Hegner 2004), we have suggested in (Johnson *et al.* 2002) that the syntax (or database schema) be specified by a certain type of mixed sketch, and that the semantics (or database states) be the category of models of the sketch. Several other authors have adopted variants of this idea, notably Diskin and Cadish in (Diskin and Cadish 1995) and Piessens and Steegman in (Piessens and Steegmans 1995). Using sketches for syntax provides a natural way to define a view, namely as a morphism of sketches. The view definition mapping is then the induced (substitution) functor between the model categories. With this formalism, an update of a single view state (an insertion or a deletion) is a morphism in the category of view states. In that case a criterion for updatability is the existence of an (op)cartesian arrow in the database state category. Note that when states can be compared in some way (so that the database states have more structure than an abstract set), there is also a natural requirement for a comparison morphism between a (view) state and its value under an update process.

When a view definition (substitution) functor is a lens in **cat**, the category of categories, it is a projection and hence also a fibration and an opfibration. Thus all (delete or insert) view updates for the view have a best possible database update. Though projections are certainly among the updatable view definition mappings, the main results of this article show that our fibrational criteria are sufficient to guarantee the existence of "universal translations". This result arises as follows. In the categorical data model, it is reasonable to modify the lens concept so that the domain of the Put is a suitable comma category, and rewrite the lens equations there. We call the resulting concept a *c-lens*. We show that a c-lens is nothing other than an opfibration. This characterization allows the application of facts about opfibrations to provide a universal translation for a functorial update process.

Acknowledgement: The authors are happy to acknowledge helpful suggestions from the referees.

2. Sketches, views and translations

We assume the reader is familiar with sketches and their models as described, for example, in (Barr and Wells 1995) and begin by defining EA sketches, their views, "propagatability" for view updates, and then the fibrational criterion for propagatability.

EA sketches are mixed sketches with limitations on cones and cocones. The important point is that the permitted cones and cocones are sufficient to implement the fundamental database operations, but they are restricted enough to permit construction of the query language. To set notation, we begin with the definition of sketch and then specialize to EA sketches.

Definition 2.1. A *sketch* $\mathbb{E} = (G, \mathbf{D}, \mathcal{L}, \mathcal{C})$ consists of a directed graph G, a set $\mathbf{D}$ of pairs of paths in G with common source and target (called the *commutative diagrams*) and sets of cones $\mathcal{L}$ and cocones $\mathcal{C}$ in G. The category generated by the graph G with commutative diagrams $\mathbf{D}$ is denoted $C(\mathbb{E})$. An *EA sketch* $\mathbb{E}$ has only finite cones and finite discrete cocones and has a specified cone with empty base whose vertex is called 1. Edges with domain 1 are called *elements.* The vertex of a discrete cocone all of whose injections are elements is called an *attribute*. A node of G which is neither an attribute nor 1 is called an *entity*.

Example 2.1. As an example data domain we consider an EA sketch for part of a movies database.

The nodes of the graph G will include persons, for example Actors, Directors and Crew, and the movies themselves. Other nodes of the graph will tabulate relationships among people and movies, for example directors direct movies, actors play in a movie, and so on.

Among the constraints will be requirements that the general Person node be a sum of Actors, Directors and Crew (as a consequence an actor cannot be a director or crew). Finite limit constraints will include expressions of joins: for example Comedy directors are found as the join (pullback) of the instances of the Directs node with Comedies. Subset constraints can also be expressed, for example, a comedy is a movie.

The EA sketch might look something like Figure 1. The graphic was produced using the EASIK (Rosebrugh *et al.* 2009) implementation for sketches; some attributes are shown UML style; constraints are noted with dashed links; monic constraints are diamond-tailed arrows.

In principle a model for a sketch may take values in any category. As we are interested in databases, we consider only models in finite sets $\mathbf{set}_f$:

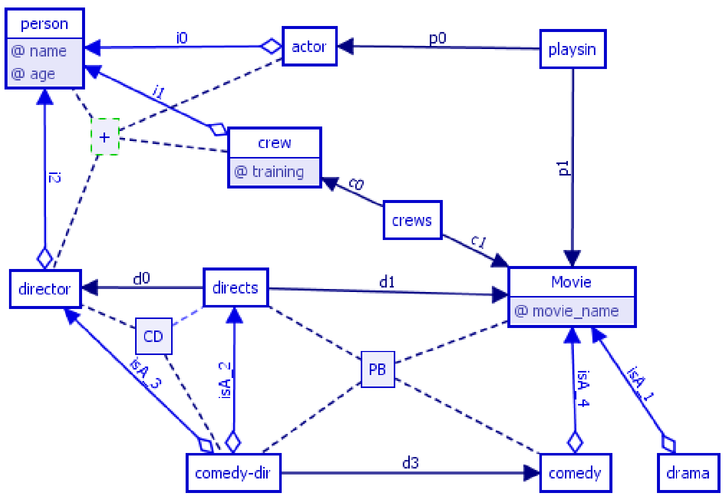


Fig. 1. Part of a movie database sketch

Definition 2.2. A *model* M of a sketch $\mathbb{E}$ is a functor $M : C(\mathbb{E}) \to \mathbf{set}_f$ such that the image of a cone in $\mathcal{L}$ (cocone in $\mathcal{C}$) is a limit cone (colimit cocone) in $\mathbf{set}_f$. If M and M' are models of $\mathbb{E}$ a *morphism* $\varphi : M \to M'$ is a natural transformation from M to M'. The category $\mathrm{Mod}(\mathbb{E})$ has as objects the models of $\mathbb{E}$ and as arrows the morphisms of models. For an EA sketch, a model is called a *database state* and we abuse notation by writing $D : \mathbb{E} \longrightarrow \mathbf{set}_f$.

To define views for an EA sketch, we use the *query language*. For an EA sketch $\mathbb{E}$ there is a category called the *theory* of $\mathbb{E}$ denoted $Q\mathbb{E}$. This $Q\mathbb{E}$ is constructed by starting from $C(\mathbb{E})$ and then formally adding all finite limits and finite sums, subject to the (co)cones in $\mathcal{L}$ and $\mathcal{C}$ (for details consult (Barr and Wells 1985, Section 8.2)). Thus $Q\mathbb{E}$ contains an object for any expression in the data for $\mathbb{E}$ constructible using finite limits and finite sums which justifies terming it the query language.

Every category has an *underlying sketch*, and the category of models $\mathrm{Mod}(Q\mathbb{E})$ of the underlying sketch of $Q\mathbb{E}$ is equivalent to the category $\mathrm{Mod}(\mathbb{E})$. Briefly, a $Q\mathbb{E}$ model restricts to an $\mathbb{E}$ model, and conversely an $\mathbb{E}$ model determines values on its queries and thus a $Q\mathbb{E}$ model.

Definition 2.3. A *view* of an EA sketch $\mathbb{E}$ is an EA sketch $\mathbb{V}$ together with a sketch morphism $V : \mathbb{V} \to Q\mathbb{E}$.

This definition allows the view (morphism) to have values which are query results in $\mathbb{E}$.

We mention without providing detail that the process Q of constructing the theory of a sketch defines a monad on a suitable category of sketches. As just defined, a view is a morphism of the Kleisli category for Q. Thus, as also noted in (Johnson and Rosebrugh 2007), views can be composed.

Via the equivalence of $\text{Mod}(\mathbb{E})$ with $\text{Mod}(Q\mathbb{E})$, a database state $D : \mathbb{E} \to \mathbf{set}_f$ may also be considered as a model $Q\mathbb{E} \to \mathbf{set}_f$, also denoted D. Composing the latter model with a view $V : \mathbb{V} \to Q\mathbb{E}$ defines a $\mathbb{V}$ database state or *view state* $DV : \mathbb{V} \to Q\mathbb{E} \to \mathbf{set}_f$, the V*-view of* D. This operation of composing with V is written V^* so $V^*D = DV$ and it defines a functor $V^* : \text{Mod}(\mathbb{E}) \to \text{Mod}(\mathbb{V})$.

Example 2.2. A view on the movies EA sketch (database schema) could be defined for the information on persons. Here a view state arises from every state of the underlying database schema by extracting only the information on persons in that state. Note that there is an EA sketch which has (in this case) a subgraph of the original graph. The view updating problem arises when we consider that a user of the persons view may wish to insert or delete information in the view state. Is there an update to the state from which the view state was derived that correctly implements the change? Is there a best possible update to that state?

In much of the literature on views and updates, a database state or a view state is merely an element of an abstract set (Bancilhon and Spyratos 1981), (Gottlob *et al.* 1988) or of a partially ordered set (Hegner 2004) rather than an object of a category. As a result, the *view definition mapping* was taken to be a (surjective) function or a monotone mapping. In the abstract set context, there is no basis for considering how one state updates to another. Rather, an update is defined only in terms of a process on the set of states, that is as an endomapping of the states. Thus, it is common to consider a view update as a process on all of the view states. We have argued that it is important to be able to consider an update of a single state. This is easy to define for a model of an EA sketch.

Definition 2.4. An *insert update* (respectively *delete update*) for a database state D is a monomorphism $D \rightarrowtail D'$ (respectively $D' \rightarrowtail D$) in $\text{Mod}(\mathbb{E})$.

The following is a criterion for being able to lift an (insert) update on a single view state to the underlying database:

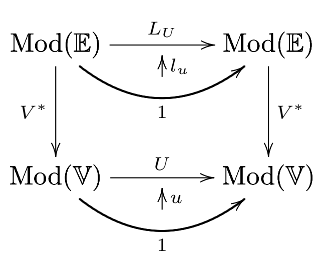
Definition 2.5. Let $V : \mathbb{V} \to Q\mathbb{E}$ be a view of $\mathbb{E}$. Suppose D is a database state for $\mathbb{E}$, and $i : V^*D \rightarrowtail W$ is an insert update of V^*D. The insertion i is *propagatable* if there exists an insert update $m : D \rightarrowtail D'$ in $\text{Mod}(\mathbb{E})$ such that $i = V^*m$ and for any database state D'' and insert update $m'' : D \rightarrowtail D''$ such that $V^*m'' = i'i$ for some $i' : W \rightarrowtail V^*D''$, there is a unique insert $m' : D' \rightarrowtail D''$ such that $V^*m' = i'$. If every insert update on V^*D is propagatable, we say that the view state V^*D is *insert updatable*.

This definition is simply a precise statement of the requirement that m be the "best" (minimal) insert update of D which maps to i under V^*. Note that Hegner (see (Hegner 2004), Lemma 4.2) states in essence that his notion of an update strategy for a closed update family (see below) provides propagatability for inserts.

To define *propagatable* for a deletion $d : W \rightarrowtail V^*D$ and *delete updatable* for a view state we simply reverse arrows. It is often the case that all arrows in $\text{Mod}(\mathbb{E})$ are monic. For example, this so if $\mathbb{E}$ is *keyed* (see (Johnson *et al.* 2002)). The reader may have noted that in that case the arrow m in Definition 2.5 is *opcartesian* and the analogous arrow for a delete is *cartesian.* In any case, it makes sense to drop the monic requirement above and generalize the notion of insert and delete update by calling any morphism of database states with domain D an insertion in D, and similarly for deletes. *We adopt this convention from here on.* When all insert (respectively, delete) updates of a view are propagatable then V^* is an *opfibration (respectively fibration)*, and conversely. Some criteria guaranteeing that V^* is an (op)fibration are discussed in (Johnson and Rosebrugh 2007).

When the database states are the category of models for an EA sketch, we can also consider an update process. It ought to be a functor. Then we can consider a "translation" of a view update process to be a compatible update process (functor) on states of the underlying database as mentioned in the Introduction. Thus compatibility requires that the view update functor and its translation commute with the view substitution V^*. However since we now have morphisms among states available, it is natural to require a comparison between a state and its image under the process (the updated state). Furthermore, we can also say when a translation is best possible, as we did above for propagatable single updates. These considerations motivate:

Definition 2.6. Let $V : \mathbb{V} \longrightarrow Q\mathbb{E}$ be a view. A *pointed view (insert) update* is a pair $\langle U, u\rangle$ where $U : \mathrm{Mod}(\mathbb{V}) \longrightarrow \mathrm{Mod}(\mathbb{V})$ is a functor and u is a natural transformation $u : 1_{\mathrm{Mod}(\mathbb{V})} \longrightarrow U$ is a natural transformation. If $\langle U, u\rangle$ is a pointed view update, a *translation* of $\langle U, u\rangle$ is a pair $\langle L_U, l_u\rangle$ where $L_U : \mathrm{Mod}(\mathbb{E}) \longrightarrow \mathrm{Mod}(\mathbb{E})$ is a functor with $UV^* = V^*L_U$, $l_u : 1_{\mathrm{Mod}(\mathbb{E})} \longrightarrow L_U$ is a natural transformation and $uV^* = V^*l_u : UV^* \longrightarrow V^*L_U$:



A translation $\langle L_U, l_u\rangle$ is *universal* when, for any other translation $k : 1_{\mathrm{Mod}(\mathbb{E})} \longrightarrow K$ and $u' : U \longrightarrow U'$ with $V^*k = u'uV^*$ (so $V^*K = U'V^*$) there is a unique transformation $k' : L_U \longrightarrow K$ such that $k = l_u k'$ and $V^*k' = u'V^*$.

Notice that a pointed (view) update provides a process and a comparison from the original state to the updated state, like an insert update. There are dual notions of *copointed view update* and *couniversal translation* which correspond to delete updates.

Example 2.3. A pointed view update on the persons view states of the movies database system might be expressed by the insertion process of adding a new actor. For *any* view state there is an updated view state with the new actor added. Notice that this will also imply an update to the persons entity. Taken together, these updates define an endofunctor on the view states. This is a pointed update because for any view state there is an insertion morphism from it to the updated view state. Together these updates form a natural transformation from the identity functor to the update functor. Notice that for states where the new actor is already present the insertion is trivial.

Clearly, a universal translation is unique up to a natural isomorphism of its functor part. As in the case of propagatability, the requirement here is not simply that there be some translation for the view update process, but further that it be optimal.

3. Lenses and "constant complements"

Our ultimate goal is to study sufficient conditions for universal translations. In this section we begin with the notion of lens and review its relation to some classical results on view updatability. Then we consider lenses in the context of our categorical data model.

We begin with the context of given sets of underlying database states S, view states V, and a view definition mapping $g : S \longrightarrow V$. Here a view update process is an endomorphism $u : V \longrightarrow V$. Informally, a lens provides a way to specify a global update process $t_u : S \longrightarrow S$ that is compatible with u *no matter which* u *is chosen.* In particular, a lens specifies, for each state s and each *updated* view state v', what the value of $t_u(s)$ should be. The lens specification depends on s, but it does not depend on the particular view update u, only on its value v' at $g(s)$.

Example 3.1. If we ignore the morphisms among states of the movies database system and treat both the database states and the view states just as sets rather than categories, we can almost imagine a lens. Given a database state, and a state of the persons view, the lens creates a database state with exactly the specified persons information, ignoring the persons information from the original database state. For this to work, of course, there must be no interaction between the persons information and the other information in the original database state. To achieve this we would have to require that the original database schema be modified.

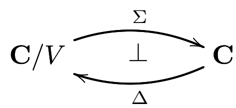
We denote projections $\pi_0 : X \times Y \longrightarrow X$ and so on, and will moreover abbreviate $\langle \pi_0, \pi_2 \rangle : X \times Y \times Z \longrightarrow X \times Z$ to $\pi_{0,2}$.

Definition 3.1. Let $\mathbf{C}$ be a category with finite products. A *lens in* $\mathbf{C}$ denoted $L = (S, V, g, p)$ has *states* S and *view states* V which are objects of $\mathbf{C}$, and two arrows of $\mathbf{C}$, a "Get" arrow $g : S \longrightarrow V$ and a "Put" arrow $p : V \times S \longrightarrow S$ satisfying the following equations:

(i) (PutGet) the Get of a Put is the projection: $gp = \pi_0$
(ii) (GetPut) the Put for a trivially updated state is trivial: $p\langle g, 1_S \rangle = 1_S$
(iii)(PutPut) composing Puts does ont depend on the first view update:
$p(1_V \times p) = p\pi_{0,2}$

The "Put" arrow does the job of specifying the database update value for pair consisting of a database state and an updated version of its image under the view mapping. "PutGet" guarantees the lifting condition mentioned above.

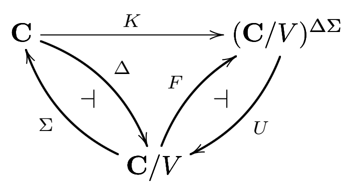
For any category $\mathbf{C}$ and any object V of $\mathbf{C}$, we denote the *slice category* by $\mathbf{C}/V$ and by $\Sigma_V : \mathbf{C}/V \longrightarrow \mathbf{C}$ the functor that remembers the domain, that is $\Sigma_V g = C$ for an object $g : C \longrightarrow V$. For $\mathbf{C}$ with finite products, the functor $\Delta_V : \mathbf{C} \longrightarrow \mathbf{C}/V$ is defined on objects by $\Delta_V C = \pi_0 : V \times C \longrightarrow V$. We will often drop the subscripts. There is an adjunction:



For an object $g : C \longrightarrow V$ of $\mathbf{C}/V$, $\Delta\Sigma g = \pi_0 : V \times C \longrightarrow V$ and the adjunction determines a monad $\Delta\Sigma$ on $\mathbf{C}/V$. The g'th component η_g of the unit for the monad is $\eta_g = \langle g, 1\rangle : C \longrightarrow V \times C$. The g'th component μ_g of the monad multiplication is $\mu_g = \pi_{0,2} : V \times V \times C \longrightarrow V \times C$.

Proposition 3.1. (Johnson *et al.* 2010) Let $\mathbf{C}$ be a category with finite products. An algebra structure on $g : C \longrightarrow V$ in $\mathbf{C}/V$ for the monad $\Delta\Sigma$ on $\mathbf{C}/V$ is determined by an arrow $p : V \times C \longrightarrow C$ satisfying the lens equations, PutGet, GetPut and PutPut, and conversely.

To explain the relation between lenses and the "constant complement" view updating strategy we consider monadicity of Δ_V. For notation, consider the following diagram in which K is the comparison functor from $\mathbf{C}$ to $\Delta\Sigma$ algebras.



The next result follows from Theorem 2.3 of (Janelidze and Tholen 1994), or it can be proved using Beck's theorem.

Proposition 3.2. (Johnson *et al.* 2010) For $\mathbf{C}$ with finite products and V an object, suppose $V \longrightarrow 1$ is split epi (so if V has a global element). Then K is an equivalence, that is, Δ is monadic.

This result means that if (S, V, g, p) is a lens, then $g : S \longrightarrow V$ is essentially just a projection to V, that is for some C, $g \cong \pi_0 : V \times C \longrightarrow V$. Indeed, given the lens (S, V, g, p), the "complement" C just mentioned is the object of $\mathbf{C}$ given by the essential inverse of K.

There is a close relationship between lenses in **set** and the "translators" of Bancilhon and Spyratos in (Bancilhon and Spyratos 1981). They define a *view* $g : S \longrightarrow V$ to be a surjective function and also define a *complete set of updates* to be a set $U \subseteq \mathbf{set}(V, V)$ of updates closed under composition and such that for u in U and s in S there is a v in U such that $vu(s) = s$. A *translator* T for U is a composition-preserving function $T : U \longrightarrow \mathbf{set}(S, S)$ such that for u in U, $gT(u) = ug$. There is a one-one correspondence between lenses and translators noted in an unpublished manuscript by B. Pierce and A. Schmitt. We note that a lens in **set** was called a "total, very-well-behaved lens" by (Bohannon *et al.* 2006).

Bancilhon and Spyratos show directly that a translator determines a product decomposition of the domain S of the view mapping and that the view mapping is the projection to the factor V. The other factor (with its projection) is called a *complementary view.*

In 2004 Hegner extended the ideas of Bancilhon and Spyratos to the ordered setting in (Hegner 2004). His idea is that the database states should be an ordered set S and that a view definition mapping should be a surjective monotone mapping $g : S \longrightarrow V$. This idea has the appealing advantage that states can be compared if they are related by the order. Hegner considers an (order-compatible) equivalence relation on the view states V. The intention is that equivalent states are mutually updatable. He defines an *update strategy* to be a (partial) mapping $p : V \times S \longrightarrow V$ satisfying a list of equations that includes both the requirement that p be monotone and the lens equations. We denote the category of partially ordered sets and monotone mappings by **pos**. It has finite limits. The present authors showed in (Johnson *et al.* 2010) that, at least for the "all" equivalence relation, an update strategy is exactly a lens in **pos**. Consequently, an update strategy or a lens provides a product decomposition of the database states for a view in **pos** (as Hegner also pointed out).

The lens concept explains the constant complement view updating strategy when database states are considered to be an abstract set or an ordered set. A $\Delta\Sigma$ algebra or a lens is the same thing as a projection to the view states. Moreover, the second factor in the projection is provided by the inverse of the equivalence K and is the "constant complement" found directly by Bancilhon and Spyratos, and also by Hegner.

The category of categories **cat** has finite products. We are interested in the view definition functor $V^* : \mathrm{Mod}(\mathbb{E}) \longrightarrow \mathrm{Mod}(\mathbb{V})$ for EA sketches $\mathbb{E}$ and $\mathbb{V}$. Whenever $\mathbb{V}$ has at least one model, so that $\mathrm{Mod}(\mathbb{V})$ has at least one object, it has a global section in **cat**. By Proposition 3.2, a lens in **cat** with view states $\mathrm{Mod}(\mathbb{V})$ is essentially a projection to its codomain. Thus whenever V^* is a lens, $\mathrm{Mod}(\mathbb{E})$ decomposes as $\mathrm{Mod}(\mathbb{V}) \times \mathbf{E}'$.

Proposition 3.3. ((Borceux 1994) 8.1.13) Let $P : \mathbf{V} \times \mathbf{C} \longrightarrow \mathbf{V}$ be a projection in **cat**. Then P is a fibration and an opfibration.

Thus, if a view definition functor V^* has a lens structure then we have insert and delete updatability of view updates. Indeed a lens structure is a powerful condition on V^* since it prescribes a view update strategy not just for updates of a single view, but also, as we will see below, for any pointed functorial update process.

Remark 3.1. When $\mathbf{C}$ has pullbacks and a terminal object (so also finite products), there is a "relative" version of the lens notion and the lens equations that we leave to the reader: for an object $\alpha : J \longrightarrow I$ in $\mathbf{C}/I$, an I-indexed family of lenses with view states α is a $\Delta_\alpha\Sigma_\alpha$-algebra structure on α. When α is split epi, then here also Δ_α is monadic.

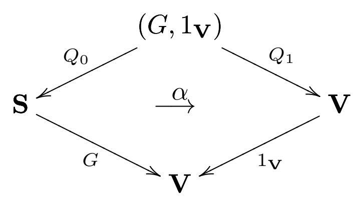
4. Fibrations and universal translations

In this section we consider a variation on the lens concept that turns out to be equivalent to being a split (op)fibration and which guarantees existence of universal translations.

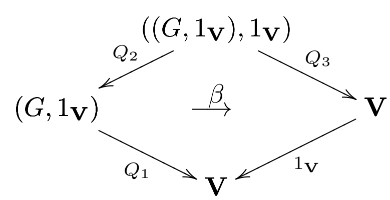
Consider again a lens (S, V, g, p) in **set**. To define a translation for an update $u : V \longrightarrow V$, it is sufficient for the Put mapping p to be defined on the subset $d_u = \{(ug(s), s) \mid s \in S\}$ of $V \times S$. As u varies, the union of the d_u is all of $V \times S$ so the domain of p should be $V \times S$. In the categorical model for a view definition $V^* : \mathrm{Mod}(\mathbb{E}) \longrightarrow \mathrm{Mod}(\mathbb{V})$, a database state D, a pointed view update $u : 1 \longrightarrow U$, and $u_D : V^*D \longrightarrow UV^*D$, to define a translation will require that we have an arrow $l_{u_D} : D \longrightarrow D'$ with $V^* l_{u_D} = u_D$. Thus, the domain of Put needs to include the arrows $u_D : V^*D \longrightarrow UV^*D$. These are arrows of the form $V^*D \longrightarrow W$ in $\mathrm{Mod}(\mathbb{V})$ so they are objects of the comma category $(V^*, 1_{\mathrm{Mod}(\mathbb{V})})$.

Example 4.1. Returning to the movies database schema and persons view, we note that for an insert update process on the view states, we are considering insertions into the image under the view definition of a database state. These insertion arrows are actually objects of the comma category just described. If we want to provide compatible updates to the original database states it is these comma category objects which must be considered.

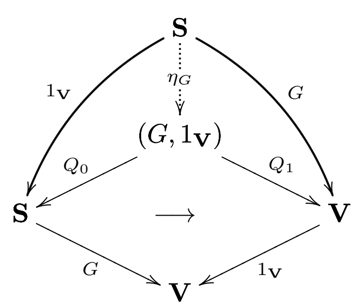
While our interest is primarily view definition functors V^*, the following definitions and results make sense for any functor $G : \mathbf{S} \longrightarrow \mathbf{V}$. For notation we denote the comma category and projections for a functor G, as follows:



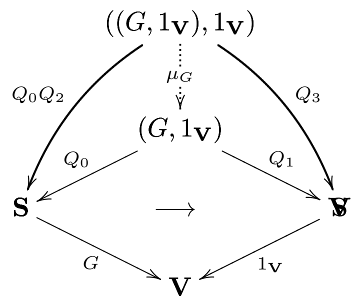
and recall that a functor $\mathbf{X} \longrightarrow (G, 1_{\mathbf{V}})$ is specified by a triple (H, K, φ) where $H : \mathbf{X} \longrightarrow \mathbf{S}$, $K : \mathbf{X} \longrightarrow \mathbf{V}$ and $\varphi : GH \longrightarrow K$. Using this we establish some further notation. First we denote the iterated comma category



Then define a functor $\eta_G = (1_{\mathbf{V}}, G, 1_G) : \mathbf{S} \longrightarrow (G, 1_{\mathbf{V}})$ as in



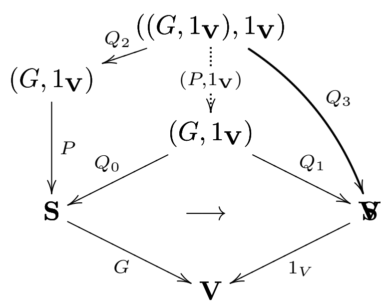
and define $\mu_G = (Q_0Q_2, Q_3, \beta(\alpha Q_2)) : ((G, 1_{\mathbf{V}}), 1_{\mathbf{V}}) \longrightarrow (G, 1_{\mathbf{V}})$ where we recall that $\beta(\alpha Q_2) : GQ_0Q_2 \longrightarrow Q_1Q_2 \longrightarrow Q_3$ in



And finally, for a functor $P : (G, 1_{\mathbf{V}}) \longrightarrow \mathbf{S}$ satisfying $GP = Q_1$ so that $GPQ_2 = \beta : Q_1Q_2 \longrightarrow Q_3$, define, for use in Definition 4.1,

$$(P, 1_{\mathbf{V}}) = (PQ_2, Q_3, \beta) : ((G, 1_{\mathbf{V}}), 1_{\mathbf{V}}) \longrightarrow (G, 1_{\mathbf{V}})$$

as in

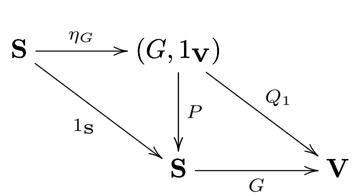


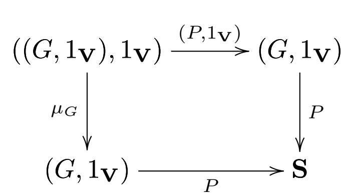
As noted above, in the **cat** case the domain of Put for insert updates should be a comma category. Notice that the equations in the next definition are similar to those of Definition 3.1 with the domain of Put replaced by the appropriate comma category.

Definition 4.1. A *c-lens* in **cat** is $L = (\mathbf{S}, \mathbf{V}, G, P)$ where $G : \mathbf{S} \longrightarrow \mathbf{V}$ and $P : (G, 1_{\mathbf{V}}) \longrightarrow \mathbf{S}$ satisfy

i) PutGet: $GP = Q_1$
ii) GetPut: $P\eta_G = 1_{\mathbf{S}}$
iii) PutPut: $P\mu_G = P(P, 1_{\mathbf{V}})$

or diagrammatically:





A c'-*lens* is as above except that the domain of P is $(1_{\mathbf{V}}, G)$, and so on.

We recall from (Street 1974) that the assignment $G \mapsto (G, 1_{\mathbf{V}})$ is the object part of a KZ monad on $\mathbf{cat}/\mathbf{V}$. The η_G and μ_G defined above provide the unit and multiplication for the monad. Dually, $G \mapsto (1_{\mathbf{V}}, G)$ is a co-KZ monad.

Proposition 4.1. An algebra structure on $G : \mathbf{S} \longrightarrow \mathbf{V}$ in $\mathbf{cat}/\mathbf{V}$ for the monad

$$\mathbf{cat}/\mathbf{V} \xrightarrow{(-,1_{\mathbf{V}})} \mathbf{cat}/\mathbf{V}$$

is determined by an arrow $P : (G, 1_{\mathbf{V}}) \longrightarrow \mathbf{S}$ satisfying the c-lens equations, PutGet, GetPut and PutPut, and conversely.

Proof. This result is similar to Proposition 3.1. We first note that the GetPut equation says that P is a morphism of $\mathbf{cat}/\mathbf{V}$ from Q_1 to G. The upper right triangle defining η_G shows that η_G is a morphism of $\mathbf{cat}/\mathbf{V}$ and the PutGet equation is then exactly the unit law for the algebra structure. Finally, the upper right triangle defining $(P, 1_{\mathbf{V}})$ is the arrow of $\mathbf{cat}/\mathbf{V}$ which defines the action of the monad applied to Q_1. Thus the PutPut equation expresses the other law required to make a c-lens an algebra. Conversely, an algebra structure on $G : \mathbf{S} \longrightarrow \mathbf{V}$ is first of all an arrow P from Q_1 to G in $\mathbf{cat}/\mathbf{V}$, GetPut is satisfied, and satisfaction of the algebra equations immediately imply satisfaction of PutGet and PutPut. □

Remark 4.1. As proved in (Street 1974), the algebras for $(-, 1_{\mathbf{V}})$ are the *split* opfibrations (and algebras for $(1_{\mathbf{V}}, -)$ are split fibrations). Of course it is also the case that pseudo-algebras for the monad in question are not-necessarily split fibrations. We could have considered those by requiring the c-lens equations *GetPut* and *PutPut* hold only up to isomorphism (equation *PutGet* would still be required). The extra generality would buy us little in the applications we have in mind, and moreover a lens in **cat** is a split fibration. This is a good place make two further points. First, to be a c-lens is to be an op-fibration. This is a *property* of a functor, not extra structure and so the algebra structure P noted above is essentially unique. That is satisfying to note because it means that there is no choice in the update strategy associated with a view functor that is a c-lens. Second, we point out that in database practice mere isomorphism of models of an EA sketch is not always a useful concept. For example, the value of an entity or attribute under a model is a particular set. While an isomorphic set might be the value in an isomorphic model, an isomorphism between Actor sets {`Harlow, Monroe`} and {`Gable, McQueen`} vastly changes the meaning of the model.

For the rest of this article, we assume that *all (op)fibrations mentioned are assumed to be split.*

Corollary 4.1. A c-lens with codomain $\mathbf{V}$ is an opfibration with codomain $\mathbf{V}$, and conversely.

The dual is that a c′-lens is a fibration.

We note that while a c-lens structure on a functor G is defined equationally, it has just been identified as an algebra structure for a KZ-monad. Thus to be a c-lens is a *property* of G rather than extra structure. Since opfibrations compose, it also follows that a composite of c-lenses is a c-lens.

Our interest is to apply Corollary 4.1 to show that a c-lens structure is sufficient to provide universal translations. For the rest of this section we discuss opfibrations, but remind the reader they are c-lenses. We begin with an example illustrating that there are interesting views whose states are updatable by the fibrational criterion when the view is a c-lens, but for which there is no lens structure. Thus

Example 4.2. A simple example shows that a non-trivial view may give rise to a c-lens which does not have a lens structure. For the base sketch $\mathbb{E}$ we take a single arrow specification: $A \xrightarrow{f} B$. The sketch $\mathbb{V}$ has a single node B and no other data. The view $V : \mathbb{V} \longrightarrow \mathbb{E}$ is just the obvious inclusion. The reader may think of V in particular as a view of a view on the movies database schema where, for example, A is `playsin`, B is `actor` and f is the arrow p_0. A model D for $\mathbb{E}$ is simply a mapping $DA \xrightarrow{Df} DB$ in **set**, and Mod($\mathbb{E}$) the category of arrows in **set**. A model for $\mathbb{V}$ is a set. Thus V^* specifies the codomain of Df. That is, V^* is the well-known “codomain” op-fibration (also a fibration if Mod($\mathbb{E}$) has pullbacks). Now V^* is not isomorphic to a product projection in **cat**, and hence it is not a lens.

We begin by recalling well-known lemmas showing that homming into an (op)fibration gives an (op)fibration, and that the pullback of an (op)fibration is again such.

Lemma 4.1. ((Borceux 1994), 8.1.15) Let $G : \mathbf{S} \longrightarrow \mathbf{V}$ be an (op)fibration. For any category $\mathbf{X}$, $(1_\mathbf{X}, G) : \mathbf{cat}(\mathbf{X}, \mathbf{S}) \longrightarrow \mathbf{cat}(\mathbf{X}, \mathbf{V})$ is an (op)fibration.

Lemma 4.2. ((Borceux 1994), 8.1.16) For a pullback in **cat**:

$$\begin{array}{ccc} \mathbf{E} & \xrightarrow{F'} & \mathbf{S} \\ {\scriptstyle G'}\downarrow & & \downarrow{\scriptstyle G} \\ \mathbf{B} & \xrightarrow[F]{} & \mathbf{V} \end{array}$$

if G is an (op)fibration, then G' is an (op)fibration.

The following consequence of the lemmas may be a new observation.

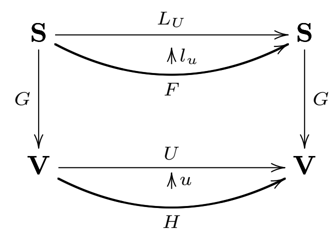
Proposition 4.2. Let $G : \mathbf{S} \longrightarrow \mathbf{V}$ be an (op)fibration and the square below be a pullback. The functor Q is an (op)fibration

$$\begin{array}{ccc} K & \longrightarrow & \mathbf{cat}(\mathbf{S}, \mathbf{S}) \\ {\scriptstyle Q}\downarrow & & \downarrow{\scriptstyle (1_\mathbf{S}, G)} \\ \mathbf{cat}(\mathbf{V}, \mathbf{V}) & \xrightarrow[(G, 1_\mathbf{V})]{} & \mathbf{cat}(\mathbf{S}, \mathbf{V}) \end{array}$$

Proof. This follows immediately from Lemmas 4.1 and 4.2. □

When Q is an opfibration, we have the following explicit description:

Corollary 4.2. Let Q in the previous theorem be an opfibration. Suppose that $HG = GF$ so that $H = Q(H, F)$ and $u : H \longrightarrow U$, then there is $L_U : \mathbf{S} \longrightarrow \mathbf{S}$ and $l_u : F \longrightarrow L_U$ such that $Gl_u = uG$ and such that for any $L' : \mathbf{S} \longrightarrow \mathbf{S}$ and $l' : F \longrightarrow L'$ satisfying $Gl' = vuG$ for some $U' : \mathbf{V} \longrightarrow \mathbf{V}$ and $v : U \longrightarrow U'$, there is a unique $k : L_U \longrightarrow L'$ with $Gk = vG$.



The important special case which we point out next clearly holds also when V^* is a lens.

Proposition 4.3. Let $V : \mathbb{V} \longrightarrow Q\mathbb{E}$ be a view and $\langle U, u\rangle$ a pointed view update. If V^* is an opfibration, then there is a universal translation $\langle L_U, l_u\rangle$ of $\langle U, u\rangle$.

Proof. Take $F = 1_{\mathrm{Mod}}(\mathbb{E})$, $H = 1_{\mathrm{Mod}}(\mathbb{V})$ in the Corollary. □

The dual is:

Corollary 4.3. Let $V : \mathbb{V} \longrightarrow Q\mathbb{E}$ be a view and $\langle U, u\rangle$ a copointed view update. If V^* is a fibration, then there is a couniversal translation $\langle L_U, l_u\rangle$ of $\langle U, u\rangle$.

Pointed view updates can be composed horizontally, and there is a comparison from a universal translation for the horizontal composite to the horizontal composite of universal translations. Formally,

Proposition 4.4. Let $V : \mathbb{V} \longrightarrow Q\mathbb{E}$ be a view and $\langle U_1, u_1\rangle$ and $\langle U_2, u_2\rangle$ be pointed view updates. If V^* is an opfibration, then $\langle U_2U_1, u_2 \circ u_1\rangle$ has a universal translation $k : 1 \longrightarrow K$ and there is a unique comparison $k' : K \longrightarrow L_{U_2}L_{U_1}$ to the composite of the (codomains of the) universal translations for $\langle U_1, u_1\rangle$ and $\langle U_2, u_2\rangle$.

Proof. This is immediate from Proposition 4.3. □

There is no reason to expect k' to be invertible, so while $\langle L_{U_2}L_{U_1}, l_{u_2} \circ l_{u_1}\rangle$ is certainly a translation for $u_2 \circ u_1$ it may not be universal.

These results show that when V^* satisfies the fibrational criteria of (Johnson and Rosebrugh 2007) for updatability, and in particular when it has a lens structure in **cat**, then universal translations are available. It is worth repeating that such translations are essentially unique, and optimal. By contrast, there is no way even to measure a translation's properties if we restrict view definition morphisms to being functions in **set**. This defect is at least partly fixed when, as in (Hegner 2004), the view definition morphism is a monotone mapping.

The following Proposition is of interest for updates in the case of a keyed EA sketch $\mathbb{E}$ so that Mod($\mathbb{E}$) is ordered. In that case it provides a partial converse to Corollary 4.2.

Proposition 4.5. Let $G : \mathbf{S} \longrightarrow \mathbf{V}$ be a functor and let the square below be a pullback.

$$\begin{array}{ccc} K & \longrightarrow & \mathbf{cat}(\mathbf{S},\mathbf{S}) \\ {\scriptstyle Q}\downarrow & & \downarrow{\scriptstyle (1_{\mathbf{S}},G)} \\ \mathbf{cat}(\mathbf{V},\mathbf{V}) & \xrightarrow[(G,1_{\mathbf{V}})]{} & \mathbf{cat}(\mathbf{S},\mathbf{V}) \end{array}$$

Suppose further that $\mathbf{S}$ is an ordered set viewed as a category and Q is an opfibration. Then the functor G is an opfibration

Proof. Suppose $\alpha : GS \longrightarrow V$ in $\mathbf{V}$. We need to define an opcartesian arrow for α. For any categories $\mathbf{A}$, $\mathbf{B}$, denote the functor which is constant at B in $\mathbf{B}$ by $K_B : \mathbf{A} \longrightarrow \mathbf{B}$. For $f : B \longrightarrow B'$, there is an obvious natural transformation $\kappa_f : K_B \longrightarrow K_{B'}$. Indeed, any natural transformation from K_B to $K_{B'}$ arises in this way.

For $\mathbf{A} = \mathbf{B} = \mathbf{V}$, denote $H = K_{GS}$ and for $\mathbf{A} = \mathbf{B} = \mathbf{S}$, $F = K_S$. Thus $HG = GF$ (so F lies over H).

Define $U = K_V : \mathbf{V} \longrightarrow \mathbf{V}$ and $u = \kappa_\alpha$ so by hypothesis there are $L_U : \mathbf{S} \longrightarrow \mathbf{S}$ and $l_u : F \longrightarrow L_U$ satisfying $Gl_u = uG$ and in particular $GL_U = UG$ so that for any S in $\mathbf{S}$, $GL_U(S) = UG(S) = V$.

Denote the arrow $l_u(S) : F(S) \longrightarrow L_U(S)$ by $\bar{\alpha} : S \longrightarrow S\alpha^*$. We aim to show that $\bar{\alpha}$ is opcartesian for α. So suppose that $\varphi : S \longrightarrow S'$ satisfies $G\varphi = \beta\alpha$, for some $\beta : V \longrightarrow GS'$.

Denote $M = K_{S'}$, and $W = K_{GS'}$ and note that $GM = WG$, $\kappa_\varphi : F \longrightarrow M$ and $\kappa_\beta : U \longrightarrow W$. Furthermore, we have $G\kappa_\varphi = \kappa_{G\varphi}G = \kappa_{\beta\alpha}G = \kappa_\beta\kappa_\alpha G = \kappa_\beta uG$.

Now since Q is an opfibration, we know that there is a unique transformation $k : L_U \longrightarrow M$ satisfying $Gk = \kappa_\beta G$ and $\kappa_\varphi = kl_u$ and we have the required fill-in arrow defined by $kS : S\alpha^* = L_U(S) \longrightarrow M(S) = S'$. Moreover, since $Gk = \kappa_\beta G$, $GkS = \kappa_\beta GS = \beta : V = U(GS) \longrightarrow W(GS) = GS'$, and necessarily, $(kS)\bar{\alpha} = \varphi$ since $\mathbf{S}$ is ordered. For the same reason, kS is unique. □

Note that we use the hypothesis that $\mathbf{S}$ is ordered to show both commutativity and uniqueness of the "fill-in" arrow. Some converse of Proposition 4.3 would be desirable, even in the case that $\mathbf{S}$ and $\mathbf{V}$ are ordered sets, but we do not know of any.

5. Conclusion and future work

The concept of lens in a category with finite products is relevant to the lifting problem also known as the view update problem for databases. Because a lens determines a product structure (up to isomorphism) on its domain, it is strong enough to guarantee that compatible liftings (or translations) can be computed for any update process. As such, it unifies interpretations of database states and view mappings in the category of sets and the category of ordered sets. The definition applies to the category of categories as well.

There the product decomposition implies that the view definition functor underlying the lens is both a fibration and an opfibration.

In (Johnson and Rosebrugh 2007), the authors have previously considered the view update problem in the context where database states are models of sketches. For a single insert update of a view state viewed as an arrow in the category of view states, the existence of an opcartesian arrow is a suitable criterion for a universal solution to the view update problem. Thus, when the view definition functor is an opfibration such problems have a solution.

In this article the focus is on update *processes* in the categorical context. That motivates considering updates to be functors. Asking for a (natural) comparison from (or to) the current state to (or from) the updated state introduces a (co)pointing of the update functor. This article shows that an obvious slight weakening of the lens concept called the c-lens is equivalent to the view functor being an opfibration (or fibration). Further, a c-lens structure on a view is sufficient to guarantee even universal updates for pointed update processes. The original lenses provide an important special case.

While a lens in the category of categories provides updates for both delete and insert functorial update processes, a c-lens structure does so only for inserts. The next step is to consider what structure on a view mapping will provide universal updating for both inserts and deletes. We expect that the categorical notion of distributive law will play a role in this study.

References

Bancilhon, F. and Spyratos, N. (1981) Update semantics of relational views, *ACM Trans. Database Syst.* **6**, 557–575.

Barr, M. and Wells, C. (1995) *Category theory for computing science.* Prentice-Hall, second edition.

Barr, M. and Wells, C. (1985) *Toposes, Triples and Theories.* Grundlehren Math. Wiss. 278, Springer Verlag.

Bohannon, A., Vaughan, J. and Pierce, B. (2006) Relational Lenses: A language for updatable views. *Proceedings of Principles of Database Systems (PODS) 2006.*

Borceux, F. (1994) *Handbook of Categorical Algebra, Vol 2.* Cambridge University Press.

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

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**.

Gottlob, G., Paolini, P. and Zicari, R. (1988) Properties and update semantics of consistent views, *ACM Trans. Database Syst.* **13**, 486–524.

Hegner, S. J. (2004) An order-based theory of updates for closed database views. *Ann. Math. Artif. Intell.* **40**, 63–125.

Hofmann, M. and Pierce, B. (1995) Positive subtyping. *SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL)*, 186-197.

Janelidze, G. and Tholen, W. (1994) Facets of Descent I. *Appl. Categ. Structures* **2**, 245–281.

Johnson, M. and Rosebrugh, R. (2007) Fibrations and universal view updatability. *Theoret. Comput. Sci.* **388**, 109–129.

Johnson, M., Rosebrugh, R. and Wood, R. J. (2002) Entity-relationship-attribute designs and sketches. *Theory Appl. Categ.* **10**, 94–112.

Johnson, M., Rosebrugh, R. and Wood, R. J. (2010) Algebras and Update Strategies. *J.UCS* **16**, 729–748.

O'Hearn, P. and Tennent, R. (1995) Parametricity and local variables. *J.ACM* **42**, 658–709.

Oles, F. J. (1982) A category-theoretic approach to the semantics of programming languages. PhD Thesis, Syracuse University.

Oles, F. J. (1986) Type algebras, functor categories and block structure. In *Algebraic methods in semantics*, 543–573. Cambridge University Press.

Piessens, F. and Steegmans, E. (1995) Categorical data specifications. *Theory Appl. Categ.* **1**, 156–173.

Rosebrugh, R., Fletcher, R., Ranieri, V., Green, K., Rhinelander, J., and Wood, A. (2009) EASIK: An EA-Sketch Implementation Kit. Available from `http://www.mta.ca/~rrosebru`, accessed 2010 August 20.

Street, R. (1974) Fibrations and Yoneda's lemma in a 2-category, Lecture Notes in Math. 420, 104–133.